NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

USING NETWORK MANAGEMENT SYSTEMS TO DETECT DISTRIBUTED DENIAL OF SERVICE ATTACKS

by

Chandan Singh Negi

September 2001

Thesis Advisor:	Alex Bordetsky
Associate Advisor:	Paul Clark

Report Documentation Page

Report Date 30 Sep 2001	**Report Type** N/A	**Dates Covered (from... to)** -

Title and Subtitle Using Network Management Systems To Detect Distributed Denial Of Service Attacks	**Contract Number**
	Grant Number
	Program Element Number
Author(s) Chandan Singh Negi	**Project Number**
	Task Number
	Work Unit Number
Performing Organization Name(s) and Address(es) Research Office Naval Postgraduate School Monterey, Ca 93943-5138	**Performing Organization Report Number**
Sponsoring/Monitoring Agency Name(s) and Address(es)	**Sponsor/Monitor's Acronym(s)**
	Sponsor/Monitor's Report Number(s)
Distribution/Availability Statement Approved for public release, distribution unlimited	
Supplementary Notes	
Abstract	
Subject Terms	
Report Classification unclassified	**Classification of this page** unclassified
Classification of Abstract unclassified	**Limitation of Abstract** UU
Number of Pages 137	

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2001	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Title (Mix case letters) Using Network Management Systems To Detect Distributed Denial Of Service Attacks			**5. FUNDING NUMBERS**
6. AUTHOR(S) Chandan Singh Negi			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Distribution Statement (mix case letters)PUT DISTRIBUTION STATEMENT HERE			**12b. DISTRIBUTION CODE**

13. ABSTRACT *(maximum 200 words)*

Distributed Denial of Service (DDoS) Attacks have been increasingly found to be affecting the normal functioning of organizations causing billions of dollars of losses. Organizations are trying their best to minimize their losses from these systems. However, most of the organizations widely use the Network Management Systems (NMS) to observe and manage their networks. One of the major functional areas of a NMS is Security Management. This thesis examines how the Network Management Systems could aid in the detection of the DDoS attacks so that the losses from these could be minimized. The thesis details the SNMP MIB variables of importance for detecting these attacks and the MIB signatures of the specific attack.

14. SUBJECT TERMS Intrusion Detection, Network Management, Security Management, Denial Of Service, Distributed Denial Of Service Attacks, Trinoo, TFN, TFN2K			**15. NUMBER OF PAGES** 137
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

USING NETWORK MANAGEMENT SYSTEMS TO DETECT DISTRIBUTED DENIAL OF SERVICE ATTACKS

Chandan Singh Negi
Lieutenant, Indian Navy
B.Tech.(Electrical Engineering),
Naval College of Engineering, INS Shivaji, Lonavla
Jawaharlal Nehru University, India, 1994

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS TECHNOLOGY
AND
MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2001

Author: [illegible]
Chandan Singh Negi

Approved by: [illegible]
Alex Bordetsky, Thesis Advisor

Paul C. Clark
Paul Clark, Associate Advisor

[illegible]
Dan Boger, Chairman
Information Systems Academic Group

[illegible]
Christopher S. Eagle, Chairman
Computer Science Department

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Distributed Denial of Service (DDoS) Attacks have been increasingly found to be affecting the normal functioning of organizations causing billions of dollars of losses. Organizations are trying their best to minimize their losses from these systems. However, most of the organizations widely use the Network Management Systems (NMS) to observe and manage their networks. One of the major functional areas of a NMS is Security Management. This thesis examines how the Network Management Systems could aid in the detection of the DDoS attacks so that the losses from these could be minimized. The thesis details the SNMP MIB variables of importance for detecting these attacks and the MIB signatures of the specific attack.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGEMENTS

I would first like to thank my parents and my family members for their unconditional love and affection throughout my life, which has given me the inner strength to complete this thesis.

I would also like to express my thanks to my advisors Professor Alex Bordetsky and Paul Clark for their guidance and help. Special thanks goes to Paul Clark for also providing me the resources to complete my experimentation.

And lastly I would like to express my gratitude and indebtedness to my teachers at various stages of my life who imparted fundamental knowledge to me, which in no less part enabled the timely completion of this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. VULNERABILITIES

The widespread usage of Internet and networking, whilst increasing the productivity, efficiency and knowledge sharing has resulted in additional problems in the hands of the computer security personnel. The increase in the vulnerability of the systems connected to the Internet is not only due to the fact that more systems are available for the attack, but also due to the fact that more systems are available from which the attack could be carried out. Also, the advancement in technology has provided sophisticated attack tools, which can be used even by people not having much competence. Hence, a day probably does not pass without some sort of compromise in the private networks.

Even though each operating system usually comes with their own implementations of Discretionary and/or Mandatory access control mechanisms, experience has shown that intruders have often been able to compromise these systems due to a variety of reasons, the most common being human related. Also, available are third party software that sit on top of the system kernel and monitor the system activity, looking for the specific security policy violations. The degree of protection that exists on any given host depends on how much time and effort has been put into applying these mechanisms. Often times, this requires much more time and effort than is desired by the various system administrators and many systems are left in a quite open and extremely vulnerable state. Also, many of these security mechanism have a significant effect on the performance of a system and thus may not be used for this reason as well. Additional layers of security are needed for the protection and so products like the Intrusion

Detection Systems (IDS), Firewalls and Virtual Private Networks (VPNs) are being used by a lot of companies in response to the new threats.

Whilst big corporations do have a dedicated security staff to take care of the security aspect, a large number of computers connected to the Internet at any time are those of a common man who is largely ignorant of all these security aspects. With the falling prices of obtaining a Digital Subscriber Line (DSL) connection or a cable modem connection, more and more such people are having these high-speed connections with the end result that these computers form very lucrative targets for any hacker.

B. DISTRIBUTED DENIAL OF SERVICE ATTACKS

Even though much progress has been made in the field of computer security, new threats seem to be developing for the network. Some of the new threats have focused on the Denial of Service (DoS), which is characterized by an explicit attempt by an attacker to prevent legitimate users of a service from using the desired resources. Examples of DoS attack include attempting to "flood" a network thereby preventing legitimate traffic, attempts to disrupt connections between two machines thereby preventing access to a service, attempts to prevent a particular individual from accessing a service and attempts to disrupt service to a specific system or person.

In fact, a new variety of DoS attack called the Distributed Denial Of Service attack (DDoS) has been found to be increasing. The DDoS attack is a DoS attack multiplied by the number of attackers. These attacks first shot into prominence when major sites like Yahoo, Amazon.com, CNN.com and others were targeted in February 2000. The Federal Bureau of Investigation has also indicated that most of the new threats would be of the nature of DDoS attacks [NIPC-00]. The losses caused by DDoS attacks are tremendous especially to e-commerce sites. According to Jupiter Communications, 46% of consumers report that poor site performance drove them away from their preferred sites. Unacceptable download times often caused by DDoS are estimated to have caused losses of up to $4.35 billion in U.S. e-commerce sales in 1999. And worldwide businesses experienced about 3.3% of unplanned downtime in 1999, translating to $1.6 trillion in lost revenue [EHAT-00].

One of the reasons for the increase in these new types of attacks is as mentioned earlier, the ease of availability of low cost, high speed Internet access to the common man who is not very well-versed with the aspects of computer security. On the technical side, the reason for these attacks is the inherent trusting nature of the Internet. The Internet protocols were not designed with security in mind, and most of the authentication is based on the IP address, which as is well known, can be easily spoofed.

C. SECURITY MANAGEMENT WITH NETWORK MANAGEMENT SYSTEM

More and more computer systems are being networked, and distributed processing is increasing in importance. Within a given organization, the trend is towards larger, more complex networks supporting more applications and more users. As these networks grow in scale, it becomes increasingly difficult to manage them by human effort alone. The complexity of such a system necessitates the use of automated network management tools. Also, since the network of most of the organizations includes equipment from multiple vendors, managing such networks without the aid of such network management tools is nearly impossible.

As networked installations become larger, more complex and more heterogeneous, the cost of network management rises. Therefore, most of the big corporations have some sort of a Network Management System (NMS) in place. A NMS has five major functional areas, Fault Management, Accounting Management, Configuration and Name Management, Performance Management and Security Management. Because of the ever increasing nature of the security attacks and threat, Security Management (SM) is of growing interest in industry and research. Four general methods are used in the SM:

- Scanning a network to determine known security loopholes in a network.
- On-line monitoring of the network to detect any suspicious events.
- Data encryption and secure passwords.
- Firewalls and perimeter defense.

Since the Network Management System (NMS) is so widely used it would be a good idea if it could be used for detecting and preventing DDoS attacks. NMS uses the Simple Network Management Protocol (SNMP) to carry out online monitoring of the network to be able to manage it. This thesis looked into the aspect whether this online monitoring of the network by NMS could give an indication of a Distributed Denial of Service Attack (DDoS). In other words, this thesis tried to answer the question whether and if yes, how could a Network Management System be used for detecting a DDoS attack? Specifically, the thesis studied the changes in the various Management Information Base (MIB) variables, which could be used for detection the DDOS attacks. Whilst the thesis did not offer any methodology of response to the attempted Security Violation, it is however expected that the NMS will also coordinate the response to the attack. This could be an area for future work.

The rest of the thesis is organized as follows. Chapter II discusses the Network Management Systems domain and also goes into the details of the nature of distributed denial of service attacks. Chapter III gives a description of some of the work carried out by the researchers in the related area of detecting/preventing the DDoS attacks. Chapter IV details the experiments carried out for studying the various MIB variables which a network management system would have to observe for detecting a distributed denial of service attack. Chapter V discusses the results. Chapter VI offers some conclusive statements and scope for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

II. APPLICATION DOMAIN

A. NETWORK MANAGEMENT SYSTEMS

Network and distributed processing systems are of growing importance and have become critical in many areas of our life. Within a given organization, the trend is towards larger, more complex networks supporting more applications and more users. As these networks grow in scale, there are more chances of things going wrong, disabling the network or a portion of the network or degrading performance to an unacceptable level.

A large network cannot be put together and managed by human effort alone. The complexity of such a system necessitates the use of automated network management tools. A Network Management System (NMS) is a collection of tools for network monitoring and control that is integrated in the following senses [STALL].

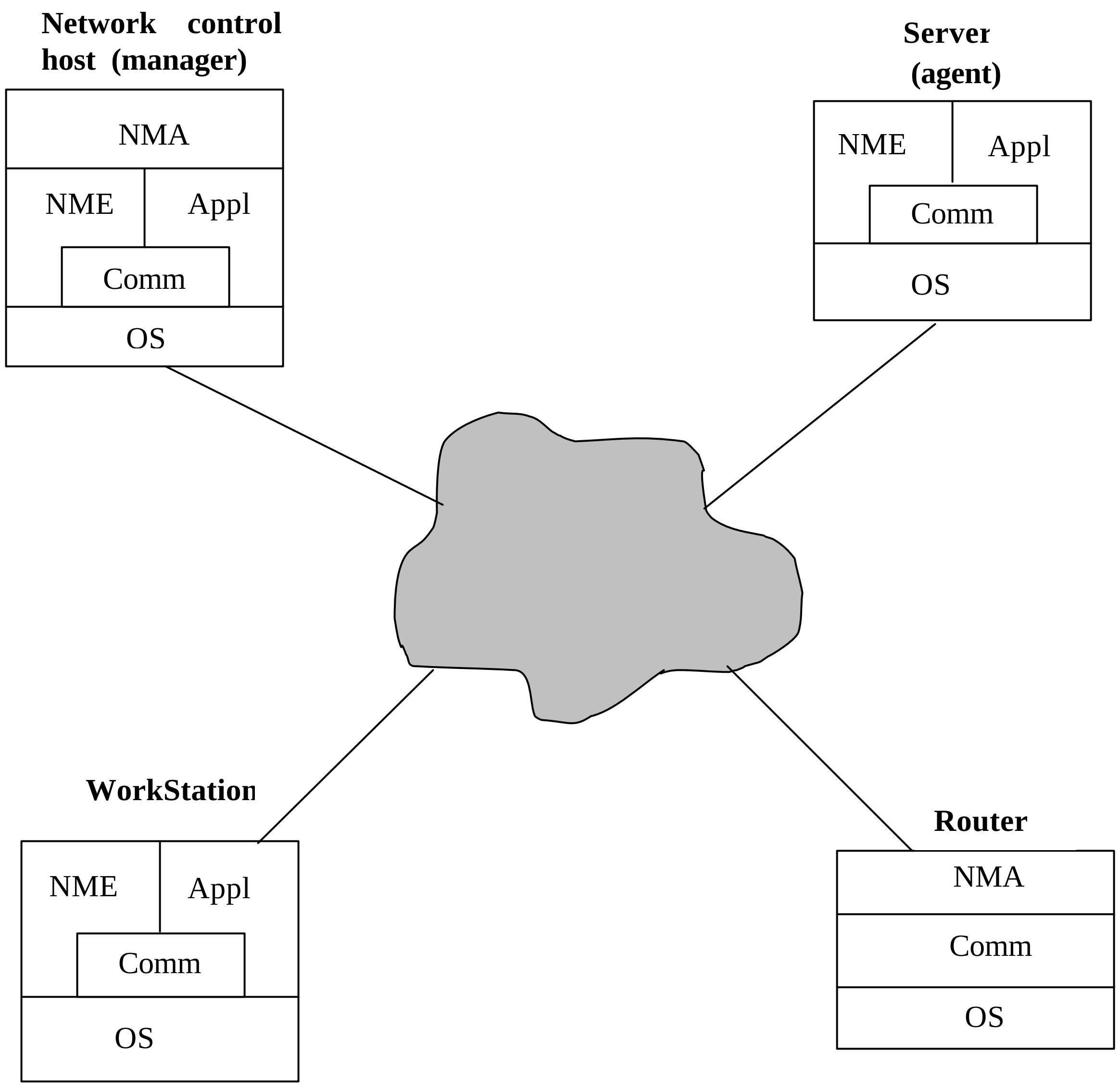
- It is a single operator interface with a powerful but user-friendly set of commands for performing most or all network management tasks.
- A minimal amount of separate equipment is necessary. That is, most of the hardware and software required for network management is incorporated into the existing user equipment.

A NMS allows network managers to view the entire network as a unified architecture, with addresses and labels assigned to each point and the specific attributes of each element and link known to the system. It consists of two active entities, the management station and the management agent. The management station is typically a stand-alone device and serves as the interface for the human network manager into the network management system. At a minimum a management station has a set of management applications for data analysis, fault recovery, and so on. It also has the capability of translating the network manager’s requirement into the actual monitoring and control of elements in the network.

The other active element in the network management system is the management agent. Key platforms such as hosts, bridges, routers, and hubs, may be equipped with the management agents so that they may be managed from the management station. The management agent responds to requests for information and actions from a management station and may asynchronously provide the management station with important but unsolicited information.

The management station and agents are linked by a network management protocol. The protocol used for the management of TCP/IP networks is the Simple Network Management Protocol (SNMP).

A general architecture of a network management system is as shown in Figure 1 [STALL].



NMA = network management application
NME = network management entity
Appl = application
Comm = communiction software
OS = operating system

Figure 1. Elements of a Network Management System. “From Ref. [STALL]”

A Network Management System performs the following five key functions:

1. Fault Management

A NMS maintains proper operation of a complex network by :

- Determining exactly where the fault is.
- Isolating the rest of the network from the failure so that it can continue to function without interference.
- Reconfiguring or modifying the network in such a way as to minimize the impact of operation without the failed component or components.
- Repairing or replacing the failed components to restore the network to its initial state.

2. Accounting Management

A NMS allows a network manager to track the use of network resources by end user or end user class. This may be for a number of reasons, including the following:

- An end user or group of end users may be abusing their access privileges and burdening the network at the expense of other end users.
- End users may be making inefficient use of the network, and the network manager can assist in changing procedures to improve performance.
- The network manager is in a better position to plan for network growth if end user activity is known in sufficient detail.
- Hence, the NMS should be able to record and provide the accounting information as desired by the network manger.

3. Configuration And Name Management

Configuration management is concerned with initializing a network and gracefully shutting down part or all of the network. It is also concerned with maintaining, adding, and updating the relationships among components and the status of components themselves during network operation. When changes in configuration occur, end users should be notified of these changes. Configuration reports can be generated either on some routine periodic basis or in response to a request for such a report. Before reconfiguration, end users often want to inquire about the upcoming status of resources and their attributes. Network managers usually want only authorized end users (operators) to manage and control network operation.

4. Performance Management

Performance management of a computer network comprises two broad functional categories – monitoring and controlling. Monitoring is the function that tracks activities on the network. The controlling function enables performance management to make adjustments to improve network performance. Some of the performance issues of concern to the network manager are:

- What is the level of capacity utilization?
- Is there excessive traffic?
- Has throughput been reduced to unacceptable levels?
- Are there bottlenecks?
- Is response time increasing?

Performance management, must therefore, monitor many resources to provide information in determining network operating level. By collecting this information, analyzing it, and then using the resultant analysis as feedback to the prescribed set of values, the network manger can become more adept at recognizing situations indicative of present or impending performance degradation. The NMS should be able to give the average and worst-case response times and the reliability of the network services. Thus, performance must be known in sufficient detail to assess specific end user queries. Performance statistics are used by the network managers to help them plan, manage and maintain large networks. Performance statistics can be used to recognize potential bottlenecks before they cause problems to the end users. Appropriate corrective action can then be taken. This action can take the form of changing routing tables to balance or redistribute traffic load during times of peak use or when a bottleneck is identified by a rapidly growing load in one area.

5. Security Management

Security management is concerned with managing information protection and access control facilities. These include generating, distributing, and storing encryption keys. Passwords, and other authorizations or access control information, must be maintained and distributed. Security management is also concerned with monitoring and controlling access to computer networks and access to all or part of the network management information obtained from the network nodes. Logs are an important security tool, and therefore, security management is very much involved with the collection, storage and examination of audit records and security logs, as well as with the enabling and disabling of these logging facilities.

Security Management keeps account of activity, or attempted activity, with these security objects in order to detect and recover from attempted or successful security attacks. This includes the following functions related to the maintenance of security information:

- Event logging
- Monitoring security audit trails
- Monitoring usage and users of security-related resources
- Reporting security violations
- Receiving notification of security violations
- Maintaining and examining security logs
- Maintaining backup copies for all or part of the security-related files
- Maintaining general network user profiles, and usage profiles for specific resources, to enable references for conformance to designated security profiles

Security management manages the access-control service by maintaining general network user profiles and usage profiles for specific resources and by setting priorities for access. The security management function enables the user to create and delete security related objects, change their attributes or state, and affect the relationships between security objects.

B. SNMP

The network of most of the large organizations includes equipment from multiple vendors, and therefore, the need for network management tools that can be used across a broad spectrum of product types, including end systems, bridges, routers, and telecommunications equipment increases. In response to this need, the simple network management protocol (SNMP) was developed to provide a tool for multi-vendor, interoperable network management [STALL]. SNMP includes the following key capabilities:

- get - enables the management station to retrieve the value of objects at the agent.
- set - enables the management station to set the value of objects at the agent.
- trap - enables an agent to notify the management station of significant events.

Resources in the network are managed by representing them as objects, which are essentially data variables that represent different aspects of the managed agent. This collection of objects is referred to as a Management Information Base (MIB). The MIB functions as a collection of access points at the agent for the management station. All of the values that are stored in the MIB are dynamic and are not stored in any file or registry key. The information stored in the MIBs ranges from Object IDs (OIDs) to Protocol Data Units (PDUs). The MIBs must be located at both the agent and the manager to work effectively.

The MIB modules have been designed by the Internet Engineering Task Force (IETF) for the standard tracks and protocol suites. Each protocol suite contains many protocols, e.g., the Internet protocol suite contains the ICMP, UDP, TCP, RIP, OSPF, EGP, and BGP protocols. A manufacturer of an equipment can come out with his/her own set of MIBs, but these need to be made available at the agent as well as at the NMS. The standard core management information is given in RFC1213.

Information stored by agents is made available to the management system by means of the following two techniques:

- Polling - Polling is a request-response interaction between a manager and agent. The manager can query any agent (for which it has authorization) and request the values of various information elements; the agent responds with information from its MIB. The request may be specific, listing one or more named variables. A request may also be in the nature of search, asking the agent to report information matching certain criteria, or to supply the manager with information about the structure of the MIB at the agent. A manager system may use polling to learn about the configuration it is managing, to obtain periodically an update of conditions, or to investigate an area in detail after being alerted to a problem.
- Event Reporting - With event reporting the initiative is with the agent, and the manager is in the role of a listener, waiting for incoming information. An agent may generate a report periodically (trap) to give the manager its current status. The reporting period may be pre-configured or set by the manager. An agent may also generate a report when a significant event (for example, a change of state) or an unusual event (for example, a fault) occurs. Event reporting is useful for detecting problems as soon as they occur. It is also more efficient than polling for monitoring objects whose states or values change relatively infrequently.

This thesis used only polling as a means to determine the changes taking place in the various variables in response to the DDoS attacks. However, there exists a possibility to modify the agents, to add some sort of intelligence in the agents, so as to raise a trap on seeing some sudden variation (as mentioned in conclusions and future works).

C. DISTRIBUTED DENIAL OF SERVICE ATTACKS

A denial of service attack is characterized by an explicit attempt by an attacker to prevent legitimate users from using resources. Examples of denial of service attacks include [STROT-00]:

- attempts to “flood” a network, thereby preventing legitimate network traffic
- attempts to disrupt connections between two machines, thereby preventing access to a service
- attempts to prevent a particular individual from accessing a service
- attempts to disrupt service to a specific system or person.

A distributed format amplifies these attacks by adding a “many to one” dimension to these attacks, making them more difficult to prevent. A distributed denial of service attack is composed of four elements, as shown in the Figure 2 [BELL –00].

- A victim - A victim is a target host that has been chosen to receive the brunt of the attack.
- Attack daemons agents - These are the agent programs that actually conduct the attack on the target victim. Attack daemons are usually deployed in host computers (slaves or zombies). These daemons affect both the target and the

host computers. The task of deploying these attack daemons requires the attacker to gain access and infiltrate the host computers.

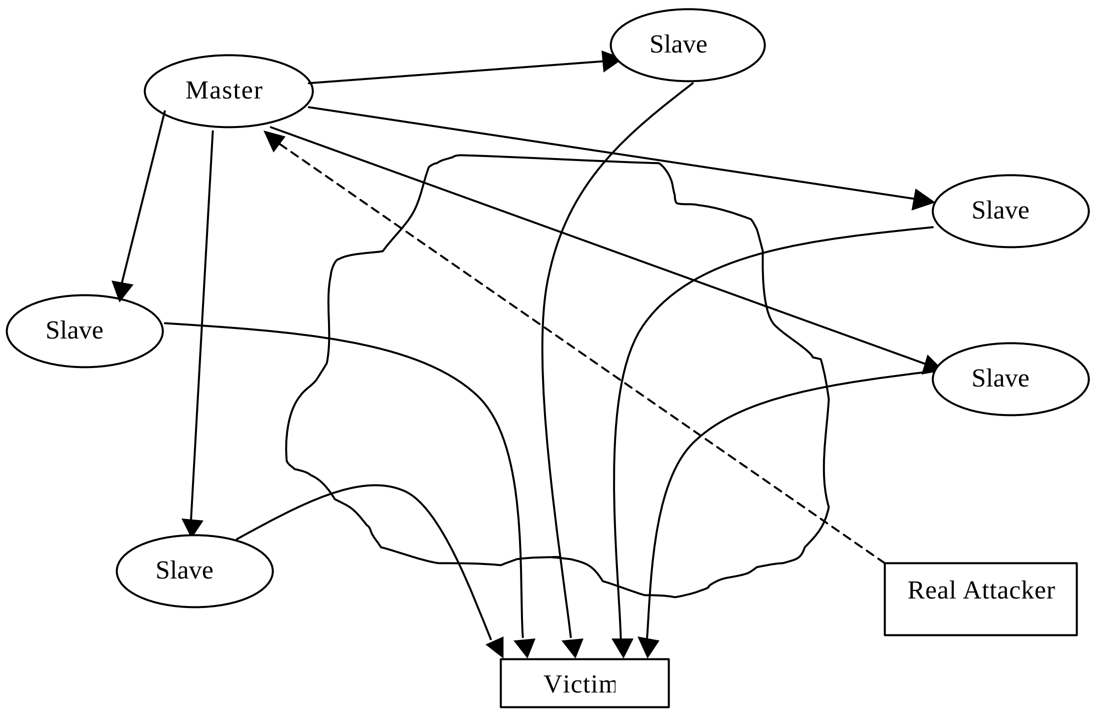


Figure 2. Components Of A Distributed Denial Of Service Attack. "From Ref. [BELL - 00]"

- Control master program - Its task is to coordinate the attack and is deployed in a host (master).
- Real attacker - The mastermind behind the attack. By using a control master program, the real attacker can stay behind the scenes of the attack [LAU -00].

The DDOS attack sequence is as follows.

- The real attacker sends an "execute" message to the control master program.

- The control master program receives the “execute” message and propagates the command to the attack daemons under its control.
- Upon receiving the attack command, the attack daemons begin the attack on the victim.

Even though it might appear that the real attacker doesn’t have much to do except to send out the “execute” command, in reality he/she actually has to plan the execution of a successful distributed denial of service attack. The attacker must infiltrate all the host computers and networks where the daemon attackers are to be deployed. The attacker must study the target’s network topology and search for the bottlenecks and vulnerabilities that can be exploited during the attack. Because of the use of attack daemons and control master programs, the real attacker is not directly involved during the attack, which makes it difficult to trace who spawned the attack.

1. Denial of Service Attack Methodology

Some of the widely known basic denial of service attack methods that are employed are as follows [STROT - 00]. These attacks take advantage of the inherent flaws in the networking protocols.

a. Bonk Attack

This attack takes advantage of the lack of bounds checking in the whole fragmentation and reassembly, by sending fragments with offsets that do not align. This makes reassembly of all the packets impossible since the positions overlap. Certain operating systems do not handle this properly and will stop further communication until the system is restarted.

b. *Ping of Death Attack*

This attack is aimed at the ICMP echo request. ICMP is used with a small payload to provide a fast, low bandwidth test of connectivity. Because of this typical usage, some software do not handle large payloads. If such software receives an ICMP request packet with a payload greater than 4000 bytes the software generates an exception and halts communication on the network.

c. *Smurf Attack/ Syn Flood*

In any TCP operation, before the handshake is complete, the receiver has no idea how long the propagation delay is between itself and the other machine. Therefore, once sending the ACK-SYN packet it will wait longer than usual (often up to 2 minutes) to receive a response. An attacker takes advantage of this and doesn't respond to the ACK-SYN packets. Thus the receiver will have to store information necessary to correlate the expected ACK response with the packet sent. There is a limited amount of space storage so if enough packets are received, without any response to the ACK-SYN packets then this space will be exhausted and the machine will cease to receive any additional SYN request. Therefore, no valid connections can be made during this period.

d. *UDP Flood*

This attack is based on UDP echo and character generator (chargen) services. Forged UDP packets are used to connect the 'echo service' on one machine to the 'chargen service' on the other machine. The result is that the two services consume all available network bandwidth between the machines as they exchange characters between themselves.

e. Land Attack

In this case the source and the destination address are the same, and so the victim consumes all its resources talking to itself, and thereby blocks outside conversations.

2. Distributed Denial of Service Attacks Methodology

In this section the various methods employed by the attackers in the various distributed of denial service (DDoS) attacks are discussed [DITT -99], [DITT -00], [BARL-00].

a. Trinoo

The communication between the master and the slaves is using TCP, whereas the communication with the attack daemons is using the UDP packets. The attack implemented is UDP flood.

b. Tribe Flood Network (TFN)

Communication between the master and the slaves is using ICMP echo reply packets. The attack could be of smurf, SYN flood, UDP flood and ICMP flood attacks.

c. TFN2K

This is the newer version of the TFN attack and it uses TCP, UDP, ICMP or all three to communicate between the master and the slaves and the communication is encrypted. The attacks implemented are the same as TFN.

d. Stacheldracht

It is based on the TFN attack. The communication between the master and the slave is encrypted and uses TCP and ICMP. In this case also the attacks implemented are the same as TFN.

e. Shaft

This attack is modeled after Trinoo. Communications between the master and the slave is achieved using the UDP packets and the attack implemented is the UDP flood attack. An important feature of this attack is its ability to switch control master servers and ports in real time, thereby making the detection by intrusion detection tools difficult.

THIS PAGE INTENTIONALLY LEFT BLANK

III. RELATED WORK

The area of the distributed denial of service attacks has been the focus of much research. Much study has been done ranging from the analysis of the various common DDoS tools available to the simulation and proposals for solving the problems. Solutions to DDoS are especially difficult because of the ability of the DDoS tools to spoof the IP address and/or the port numbers. The usage of cryptography in the attack further complicates the issues. In fact, many experts have stated that there are currently no successful defenses against a fully distributed denial of service attack. Nevertheless some of these works are discussed here, which propose some of the actions we can take to minimize the impact of the DDoS attacks.

A. IMPROVING ROUTING ALGORITHMS

Changing/improving the routing algorithms in the routers has been proposed to minimize the impact of the DDoS attacks. Felix Lau et al. simulated a DDoS attack using ns-2 simulator (a discrete event simulator used for networking research and simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks) and studied the performance of various routing algorithms during an attack in a network router. They have found that the routing algorithms like Drop tail, Fair queuing, Stochastic Fair queuing and Deficit Round Robin provide no bandwidth to the legitimate user during the attack, while Class Based queuing and Random Early Detection provides some limited bandwidth for certain classes of input flows. They have also found that implementing queuing algorithms in network routers may provide bandwidth to the users even in case of the DDoS attacks [LAU -00]. This approach however, doesn't give any indication of the attacking sources.

B. IP TRACEBACK SCHEMES

Some researchers have studied the methodology for tracing back the attacker. The basis of these studies is that the DDoS attacks are on the rise because of the difficulty in tracing back the attacking source. There have been many proposals to determine the attacking hosts in spite of usage of address spoofing techniques to disguise the source of origin. These proposals help determine the attacking hosts, not the actual attacker, because the attacker might be hiding in some other host and sends the attack command, much before the attack builds up. Some of these proposals are discussed below:

1. Packet Based Trace Back

In packet-based trace back, packets are marked with the addresses of intermediate routers. The victim can use the information inscribed in packets to trace the attack back to its source. The drawback of this method is the overhead resulting from the increase in the number of packets.

2. ICMP Trace Back Messages (itrace)

In this method the routers generate information packets – separate from the data packets – that convey analogous path information as ICMP trace back messages to the victim [BELL-00-M]. An itrace router probabilistically generates an authenticated copy of a packet, adds its own IP address as well as the IP of the previous and the next hop routers, and forwards the packet either to the source or the destination address. This approach does not need an upstream router map because the IP addresses of the routers are encoded in the itrace packets. The victim can use the information in the itrace packets to build an upstream router map. The main problem with this method also is the large overhead resulting from the additional data packets.

3. IP Trace Back using Probabilistic Packet Marking

In this approach, the routers are allowed to probabilistically mark the packets with partial path information during packet forwarding. The victim can then reconstruct the complete path after receiving a modest number of packets containing the marking [SAVA-00]. This approach has low overhead for routers and supports incremental deployment, however it has a very high computation overhead for the victim to reconstruct the attack paths. Another problem with this approach is that it has mainly considered the denial-of-service with a single attacker site. The scheme becomes inefficient and gives inaccurate results even under a minor increase in scale.

4. Advanced and Authenticated Marking Schemes for IP Trace Back

Dawn Xiaodong Song and Adrian Perrig have proposed two new schemes, the Advanced Marking Scheme and the Authenticated Marking Scheme for tracing back the approximate origin of the spoofed IP packets. This approach builds upon the previous approach to overcome its shortcomings. A new encoding scheme is used, whereby instead of the full IP address of a router, only its hash value is encoded. The schemes have low network and router overhead, and support incremental deployment. These schemes also result in lower computation overhead for the victim to reconstruct the attack paths under large-scale distributed denial-of-service attacks. Also, the authenticated marking scheme provides efficient authentication of routers' markings such that even a compromised router cannot forge or tamper markings from other un-compromised routers [SONG - 01].

C. INGRESS/ EGRESS FILTERING (RFC 2267/2827)

Ferguson and Senie proposed ingress filtering in which only valid IP packets from a valid source address are allowed by the routers to enter the Internet [FERG - 98]. Ingress traffic filtering at the periphery of Internet connected networks will reduce the effectiveness of source address spoofing denial of service attacks. The problem with this approach is that it requires the router to have sufficient power to verify the IP address of each packet and to have sufficient knowledge to distinguish between legitimate and illegitimate addresses. Also, the approach is effective only if deployed at a large scale.

Ingress filtering is basically from the viewpoint of the Internet Service Providers (ISPs) or the administrators of the Autonomous systems. The same approach from a company's internal networks view point is the egress filtering in which only valid IP source packets are allowed to leave the network (Figure 3).

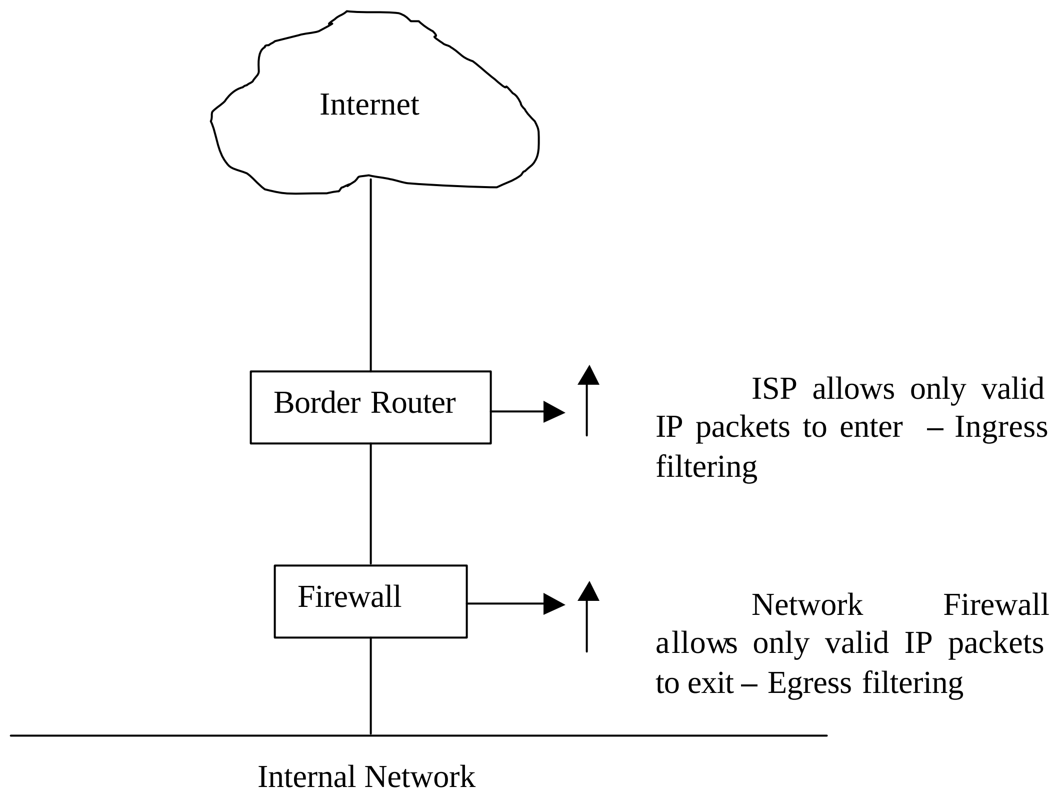

Internet
Border Router
ISP allows only valid IP packets to enter – Ingress filtering
Firewall
Network Firewall allows only valid IP packets to exit – Egress filtering
Internal Network

Figure 3. Ingress/Egress Filtering.

D. TCP INTERCEPT

Cisco has implemented a feature called TCP Intercept to protect TCP servers from TCP SYN – flooding attacks by intercepting and validating TCP connection requests [CISC -99]. It can be used in two modes. In the intercept mode, the TCP synchronization (SYN) packets from clients to servers are intercepted by the software and a connection established on behalf of the destination server. If the connection is successful then the software establishes the connection with the server on behalf of the client and knits the two half connections transparently. Thus connection attempts from unreachable hosts will never reach the server. The software continues to intercept and forward packets throughout the duration of the connection. In the case of illegitimate requests, the software goes into aggressive timeout modes for the half connections.

TCP intercept can also be used in watch mode, in which the software passively watches the connection requests flowing through the router. If a connection fails to get established in a configurable interval, the software intervenes and terminates the connection attempt.

Thus, only the valid connection requests are allowed. Illegitimate requests and half-open connections are discarded by having timeouts or resets. When under attack, the TCP intercept feature becomes more aggressive in its protective behavior.

E. PACKET FILTERING

Packet filtering is a network security mechanism for controlling what data can flow to and from network affected routers or firewalls. A firewall is a system deployed between the company’s network and the Internet and has the ability to observe and control all the incoming/outgoing packets.

A firewall can examine any packet moving between the internal network and the Internet or a firewall can be configured to serve as a proxy for communications. A firewall can limit the incoming network traffic from a particular source address thereby reducing the risk of an attack. However, since there is no guarantee that the source address is not spoofed, any filtering based on source address is subject to vulnerabilities. A firewall also provides no method of limiting valid traffic such that a machine is not simply overwhelmed with requests. But firewalls can block the type of traffic (protocols) and thus reduce the attack sets.

F. INTRUSION DETECTION SYSTEMS

Intrusion Detection Systems (IDS) monitor the network for known attack signatures. An attack signature is a sequence of events, which is known to occur prior to or during an attack. IDS have been used to prevent some of the DDoS attacks, but they have their own limitations. There is no way of knowing the attack signature until at least one site has been attacked. Moreover, the attack signature based on the port number of the attack does not seem to be very effective in light of the new developments in the field of the DDoS tools, in which the port number as well as the protocols used are random in nature. IDS, which observe traffic for attack signatures are known as knowledge-based systems.

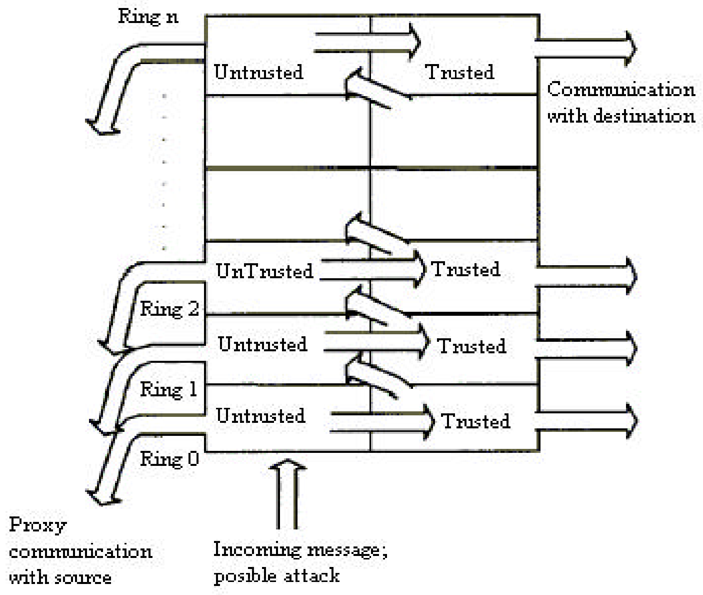
If the system has the ability to learn attack signatures, it is called a behavior based IDS. Problem with behavior-based systems is that to be effective the learning should be in real time, which is difficult to achieve. Another problem with behavior based ID systems is false positives reports. Whenever a system attempts to learn a behavior at the same time as guarding against the behavior it runs the risk of falsely identifying an occurrence. In most networks, the number of false positives must be sufficiently low so as not to disrupt normal communications.

G. ROUTE-BASED DISTRIBUTED PACKET FILTERING

In this approach, routing information is used to determine if a packet arriving at a router is valid with respect to its inscribed source/destination addresses, given the reachability constraints imposed by routing and network topology. With a partial coverage or deployment – about 20% in Internet Autonomous systems topologies – a significant fraction of the spoofed packet flows can be filtered and prevented from reaching other autonomous systems. This approach doesn’t require expanding IP header fields to encode stamped link information [PARK-00]. However, it does require compliance by a number of different autonomous systems to be effective.

H. THE NOZZLE

E. Strother has presented a protection method called a nozzle [STROT-00]. The nozzle is based upon favorable aspects of firewalls and network pumps. It is to be deployed similar to the firewall such that all conversations from an un-trusted user to a critical resource are monitored. The main advantage of the nozzle is the ability to provide a threshold for trusted traffic thus precluding new attacks. A nozzle consists of a series of rings, each of which has a trusted and un-trusted buffer, rules for packet placement, and rules for communication with the next level. Rings are placed in the protocol stack so that they can protect particular protocols. Each ring looks into the irregularities in the incoming packet for that particular protocol layer. If there is no defect then the packets are allowed to pass through either to the next ring or to the destination, depending on the configuration. A simplified diagram of the nozzle function is shown in Figure 4.



Ring n
Untrusted
Trusted
Communication with destination
UnTrusted
Trusted
Ring 2
Untrusted
Trusted
Ring 1
Untrusted
Trusted
Ring 0
Proxy communication with source
Incoming message; posible attack

Figure 4. The Nozzle. “From Ref. [STROT-00]”

The nozzle’s performance is dependent on how it is configured.

I. CENTERTRACK

CenterTrack is an overlay network, consisting of IP tunnels, that is used to selectively reroute interesting datagrams directly from edge routers to special tracking routers. It requires input debugging and IP tunnel support on edge routers and special “CenterTrack” routers. The tracking routers can easily determine the ingress edge router by observing which tunnel the datagrams arrive on. The datagrams can be examined, then dropped or forwarded to the appropriate egress point [STON-99].

THIS PAGE INTENTIONALLY LEFT BLANK

IV. EXPERIMENTAL SETUP

A. HYPOTHESIS

Joao B.D. Cabrera, Lundy Lewis et al. proposed a methodology in which a Network Management System can be used for early detection of Distributed Denial of Service attacks [JOAO – 01]. Their hypothesis is that even though there are quite large number of events that are prior to an attack (e.g. suspicious logons, start of processes, addition of new files, sudden shifts in traffic, etc.), a DDoS attack could be detected on the basis of shifts in the traffic patterns in some of the Management Information Base (MIB) variables collected from the systems participating in the attack. Using domain knowledge an experiment was carried out and it was observed by them that there are indeed MIB-based precursors of DDoS attacks, that makes it possible to detect the attack before the target is shut down. They have also described how the relevant MIB variables at the attacker can be extracted automatically using Statistical tests for Causality. That is, out of the various variables at the target and the attacker, which change over a period of time, during a DDoS attack, the main variables which need to be monitored at the attacker can be determined, which can then be used for detecting in advance, any machine participating in the attack, using a simple anomaly detection scheme of the rate of change of these key variables.

B. AIM

This thesis was phase two of the experiments suggested above. In phase one above, the MIB variables to be studied were determined by the domain knowledge of the various attacks. The phase three of the experiments would be to actually conduct attacks over two different geographical locations. The aim of this thesis was to actually test all the MIB variables, which change over a period of time during the DDoS attack in an experimental test bed and determine the key MIB variables, which should be observed for monitoring.

An offshoot of this experiment would also be to determine if the type of DDOS attack could be identified by the observed type of the changes in the MIB variables (e.g. if it is DOS SYN flood, or UDP flood etc.) so that a possibility exists of an entirely automated procedure centered on Network Management systems for detecting precursors of Distributed Denial of Service Attacks, and responding to them.

C. TESTBED

The experimental setup for the test bed consisted of a local area network as shown in Figure 5.

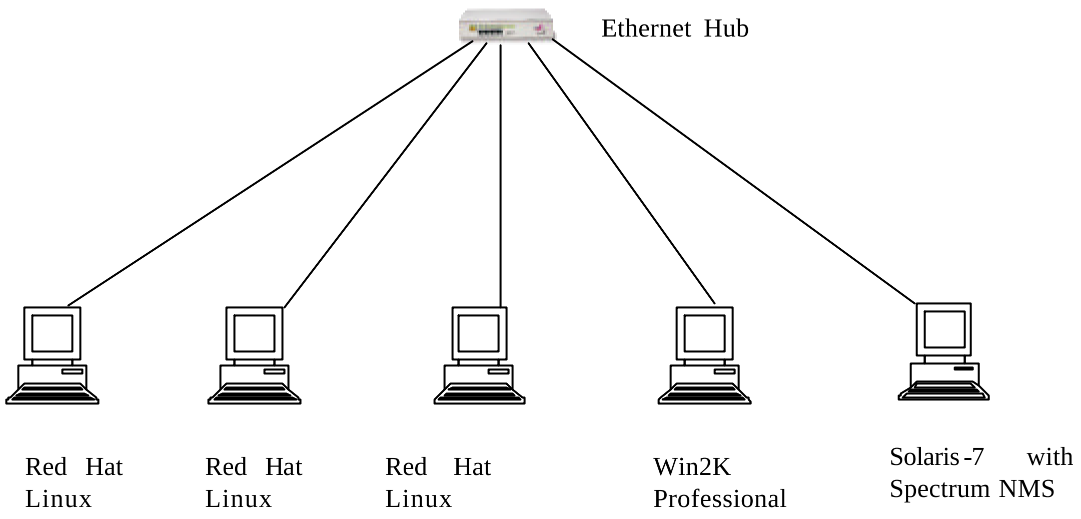


Figure 5. Experimental Setup.

The Network Management system used was the Spectrum Network Management system, manufactured by M/S Aprisma Technologies limited. The Operating system was the Sun Solaris 7.0. Two Red Hat Linux hosts were used as attackers and the target was the Windows 2000 professional system. The third Red Hat Linux host had the various programs and was used to observe the MIB variables.

The attacking hosts were inserted with the DDoS zombie codes. In reality, before carrying out the attack, the attacker would have to impregnate the attackers with the zombie code and then from a master computer would give a command to attack the target. However in this thesis the attacking hosts were assumed to have been infected already and the master computer (needed for some attacks for giving a command) was external to the setup.

D. METHODOLOGY

1. Using the Spectrum Network Management System

In the first approach the idea was to use the Spectrum Network Management System to observe the deviant variations in the various MIB variables. The java based MIB browser (which comes as a part of the Spectrum package) was used to browse the values of the various MIB variables during the duration of different attacks to shortlist the variables which undergo even a slightest change for each of the attacks. Thereafter, the target was polled for the value of each of the MIB variable but, it was observed that the NMS had the limitation of being able to poll at the time intervals of greater than or equal to 30 seconds. This was considered inadequate for getting the clear picture of the variation in the MIB variables and so the second approach of using a program ‘miblogger.c’ was resorted to.

2. Using the program 'miblogger.c'

In the second approach, a program miblogger.c was written using the ucd-snmp development library available with the Red Hat Linux and compiled and run on the same platform (the third Linux box, different from the two attackers). The list of MIBs to be queried was taken from the shortlist obtained from the first approach. The program queried the attackers and the target for the value of these MIBs at a regular interval of five seconds. The system time and the response obtained were recorded in a separate log file for each host (but containing all the shortlisted MIBs). Perl scripts were used to separate the logged MIBs into separate log files and to create the Round-Robin-Databases (RRDs) using the RRD perl module. The graphing function of the RRDtool was used to generate the graphs.

E. ROUND ROBIN DATABASE

Round Robin Database (RRD) [TOBI-00] is a system to store and display time-series data (i.e. network bandwidth, machine-room temperature, server load average). It stores the data in a very compact way that will not expand over time, and it presents useful graphs by processing the data to enforce a certain data density. It can be used either via simple wrapper scripts (from shell or Perl) or via frontends that poll network devices and put a friendly user interface on it.

RRDtool refers to the Round Robin Database tool. Round robin is a technique that works with a fixed amount of data, and a pointer to the current element. When the current data is read or written, the pointer moves to the next element. Since the database is in the form of a circular record file, there is neither beginning nor an end, one can go on and on.

After a while, when all the available places are used, the process automatically reuses old locations. This way, the database does not grow in size and therefore requires no maintenance. RRDtool works with Round Robin Databases (RRDs). It stores and retrieves data from them.

On disk, the round robin database (RRD) is organized into sequential sections, called round robin archives (RRA). Within each RRA is a section for each of the data sources (input) stored in this RRD. Each RRA is defined by a consolidation function which maps primary data points (PDP) to consolidated data points (CDP), which are then plotted. The consolidation function could be the MAX, MIN, LAST or AVERAGE of the values.

In this thesis the RRDtool is used for generating the RRD databases, updating them with the logged data and for creating graphs in the 'gif' format. The RRD perl module and simple perl scripts are used for this purpose.

F. DESCRIPTION OF THE PROGRAMS

1. Logging of MIB Variables Values

The program used to collect and log the values of the MIB variables was 'miblogger.c' in RedHat linux 7.0 (Kernel 2.2.16-22) and the GNU compiler gcc version 2.96. The program uses the UCD-SNMP library (also known as NET-SNMP). The library is freely available and also comes with the standard Red-Hat installation.

Initially load the various header files.

```
#include <ucd-snmp/ucd-snmp-config.h>
#include <ucd-snmp/ucd-snmp-includes.h>
#include <sys/time.h>
```

The polling interval is defined to be 5s and the total poll duration is for 20 minutes (20 x 60 / 5). This is a reasonable interval since the DDoS attacks result in very sudden large changes in the MIB variables.

```
#define POLL_INTERVAL  5
#define POLL_COUNT  240
```

A list of hosts and their type (whether target or attacker) is declared and populated. The type is required because the MIB variables to be observed for the attacker and the target are different. The list of the MIB variables to be observed for both the type of hosts was determined by both the domain knowledge of the attacks as well as by observing them with a MIB browser when under attack.

```
struct host { char *name; char *type; }
struct oid { char *Name; oid Oid[MAX_OID_LEN]; int OidLen;
```

The main() routine first calls the initialize() subroutine which initializes the snmp library. It also reads the various MIB variables and correlates them with their oid number, thus ensuring that they are correct, returning error if incorrect.

```
init_snmp("miblogger");
.
.
while (op->Name) {
                op->OidLen = sizeof(op->Oid)/sizeof(op->Oid[0]);
        if (!read_objid(op->Name, op->Oid, &op->OidLen)) {
                snmp_perror("read_objid");
                exit(1);
        }
}
```

The hosts are then polled ‘POLL_COUNT’ times, at ‘POLL_INTERVAL’ duration. The poll() subroutine initially declares the following variables:

- struct snmp_session – A structure that holds information about the host which is to be queried. Two of these structures need to be declared, one to fill with information and the second a pointer returned by the library.

- struct snmp_pdu – This structure holds all of the information that is sent (received) to (from) the remote host.

- struct oid – The pointer for the oid. Could be oid_T (target) or oid_A (attacker).

- FILE *f – The log file. The log file is different for each host.

The snmp-session is then initialized which prepares the input session data. The first call to snmp_sess_init() initializes the Library, including the MIB parse trees, before any SNMP sessions are created. The structure is populated with the snmp version type (v1, v2c, v3 – v2c is used in the program as it is the most common one and has wide spread support), the name of the host and the snmp community name) and a session is then setup with the call snmp_synch_setup().

```
snmp_sess_init(&ss);
ss.version = SNMP_VERSION_2c;
ss.peername = hp->name;
ss.community = "public";
snmp_synch_setup(&ss);
```

After establishing a session, it is opened. Opening it returns a pointer to another session that is used in all the future calls.

```
sp = snmp_open(&ss)
```

Next, a Protocol Data Unit (PDU) is created which is sent to the remote host for requesting the information needed. The PDU created is SNMP_PDU which retrieves a value for each of the specified oid. A NULL value needs to be added to each oid for all outgoing requests. This is achieved by the

snmp_add_null_var(req, op->Oid, op->OidLen);

Finally, the request is sent using the session pointer and the pdu. A status and a response is returned which is stored in the second snmp_pdu (*resp) structure declared earlier.

status = snmp_synch_response(sp, req, &resp);

The returned response, status, session pointer and the log file name are then passed to the log_response() subroutine for logging purpose. The returned values are logged only if there is no error (status = STAT_SUCCESS and response-> == SNMP_ERR_NOERROR). A new line is added to the log files after recording the group of MIB values at a particular time, to facilitate the parsing of the log files easily.

2. Separating the MIB Time Series

The initial log files for the two attackers and the target contains the MIB values logged over a period of time, with the latest time and the corresponding MIB value at the beginning of the file. Two short perl scripts are used for this purpose. The first perl script 'reverse.pl' is for reversing the order of the logged data. This is required for creating the RRD database. The RRD database functions only if the ordering of data values is in the increasing order of the logged time i.e. the value recorded first should be at the beginning of the file.

The script reverse.pl takes the log file and creates another file in which the data is in the reverse order. The second script is given this reversed log file as the input and it splits it into the various MIB values recorded in the log file.

Open the log file first:

```
open(LOG,"<$file") or die "could not open file";
```

Go through the file and determine the number of MIB variables in the log file, the group of MIB variables is separated in the log file by a new line character. The script looks for the first character in the line and if it is the new-line character then it exits the loop.

```
while(<LOG>) {
        if ($_ !~ /^\n/){
                chomp;
                ($time,$ipadd,$mib_n_value) = split /:/;
                print "min_n_value = $mib_n_value\n";
                ($mib,$count) = split / = /, $mib_n_value;
                ($f,$m,$l) = split /\./, $mib;
                print "The value of log file name is $m.log\n";
                sleep 1;
                $array[$no] = $m.".log";
                print "The value of $no element in array is $array[$no]\n";
                $no++;
        }
      else {last;}
}
close(LOG);
$max_no_of_mibs = $no;
```

Determine the names of the MIB variables logged in the file and then create corresponding log files for each of them.

```
for ($x = 1; $x <= $max_no_of_mibs; $x++) {
        open(OUT."$x", ">". $array[$x-1]) || die "couldn't open $x.log \n";
}
$x = 1;
```

```
open(LOG,"<$file") or die "could not open file";
while(<LOG>) {
        if ($_ !~ /^\n/) {
                chomp;
                if ($x > $max_no_of_mibs) { $x = 1; }
                ($time,$ipadd,$mib_n_value) = split /:/;
                ($mib,$count) = split / = /, $mib_n_value;

                print "Updating $x.log\n";
                $FH = OUT."$x";
                print $FH "$time:$count\n";
                $x++;
        }
}
close(LOG);
```

3. Plotting the MIB Time Series

The time series for each of the MIB variable is plotted using the RRD tool described earlier. It requires the usage of the perl::RRD module.

Use the perl RRD module.

```
use RRDs;
use strict;
```

Open the log file and determine the start time and the end time of the logging.

```
open(LOG,"<$file") or die "could not open file";
my($cnt)=1;
while(<LOG>) {
        chomp;
        ($time,$count) = split /:/;
        if ($cnt <2) { $start_time = $time-10;}
        $end_time = $time;
        $cnt++;
}
close(LOG);
```

Create and update the round-robin database for each of the MIB variable and print error if any.

```
RRDs::create ($fname.".rrd","--start",$start_time, "--step",5,
                    "DS:a:COUNTER:10:U:U",
                    "RRA:AVERAGE:0.5:1:600");
$ERROR = RRDs::error;
die "$0: unable to create '$fname.'.rrd: $ERROR\n" if $ERROR;
print "Updating $fname.rrd\n";
open(LOG,"<$file") or die "could not open file";
while(<LOG>) {
          chomp;
          RRDs::update ($fname.".rrd", $_);
          if ($ERROR = RRDs::error) {
                    die "$0: unable to update $fname.rrd: $ERROR\n";
          };
}
close(LOG);
```

Draw the graphs in gif format.

```
          RRDs::graph ("$fname.gif","--start", $start_time, "--end", $end_time,"--title",
"TFN_Syn_Flood - Rate of change of $fname with time",  "--vertical-label",
"$fname(/s)","DEF:mibvalue=$fname.rrd:a:AVERAGE",
"LINE1:mibvalue#FF0000:\rAverage change in $fname per second");
          if ($ERROR = RRDs::error) {
                    die "$0: unable to draw graph $fname-AVG.gif: $ERROR\n";
          };
```

THIS PAGE INTENTIONALLY LEFT BLANK

V. RESULTS

This chapter describes the results of the experiments carried out on the experimental setup. The results identify the affected MIB variables at the attackers and at the target for each of the DDoS attack and also graphically illustrate the change in these before during and after attack. The key variables (those which show large changes) are highlighted. The attacks considered in the experiment were the Trinoo, TFN and the TFN2K attacks. The graphical results clearly indicate the sudden abrupt change in the key MIB variables during the attack.

A. TRINOO

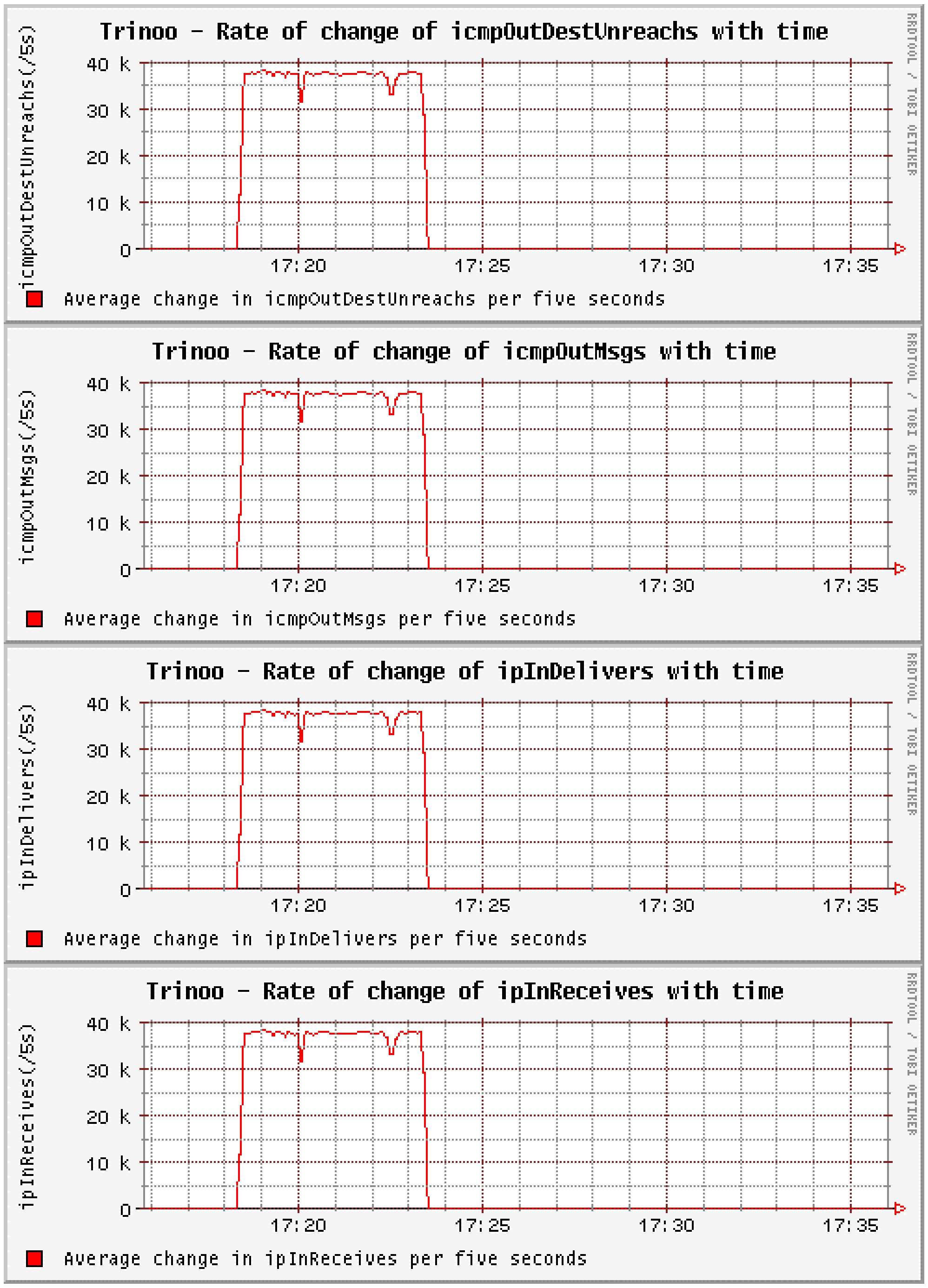
The affected variables at the target and at the attacker in a trinoo attack are shown in table 1.

Sl. No.	Target	Attacker
1.	**icmp.icmpOutDestUnreachs.0**	**icmp.icmpInDestUnreachs.0**
2.	**icmp.icmpOutMsgs.0**	**icmp.icmpInMsgs.0**
3.	**ip.ipInDelivers.0**	ip.ipInDelivers.0
4.	**ip.ipInReceives.0**	**ip.ipI nReceives.0**
5.	**ip.ipOutRequests.0**	**ip.ipOutRequests.0**
6.	udp.udpInDatagrams.0	udp.udpInDatagrams.0
7.	**udp.udpNoPorts.0**	**udp.udpOutDatagrams.0**
8.	udp.udpOutDatagrams.0	

Table 1. Affected MIB Variables In Trinoo Attack.

1. Changes in the Target MIB Variables

Figure 6 shows the average changes in the target MIB variables per five-second interval in case of a trinoo attack.



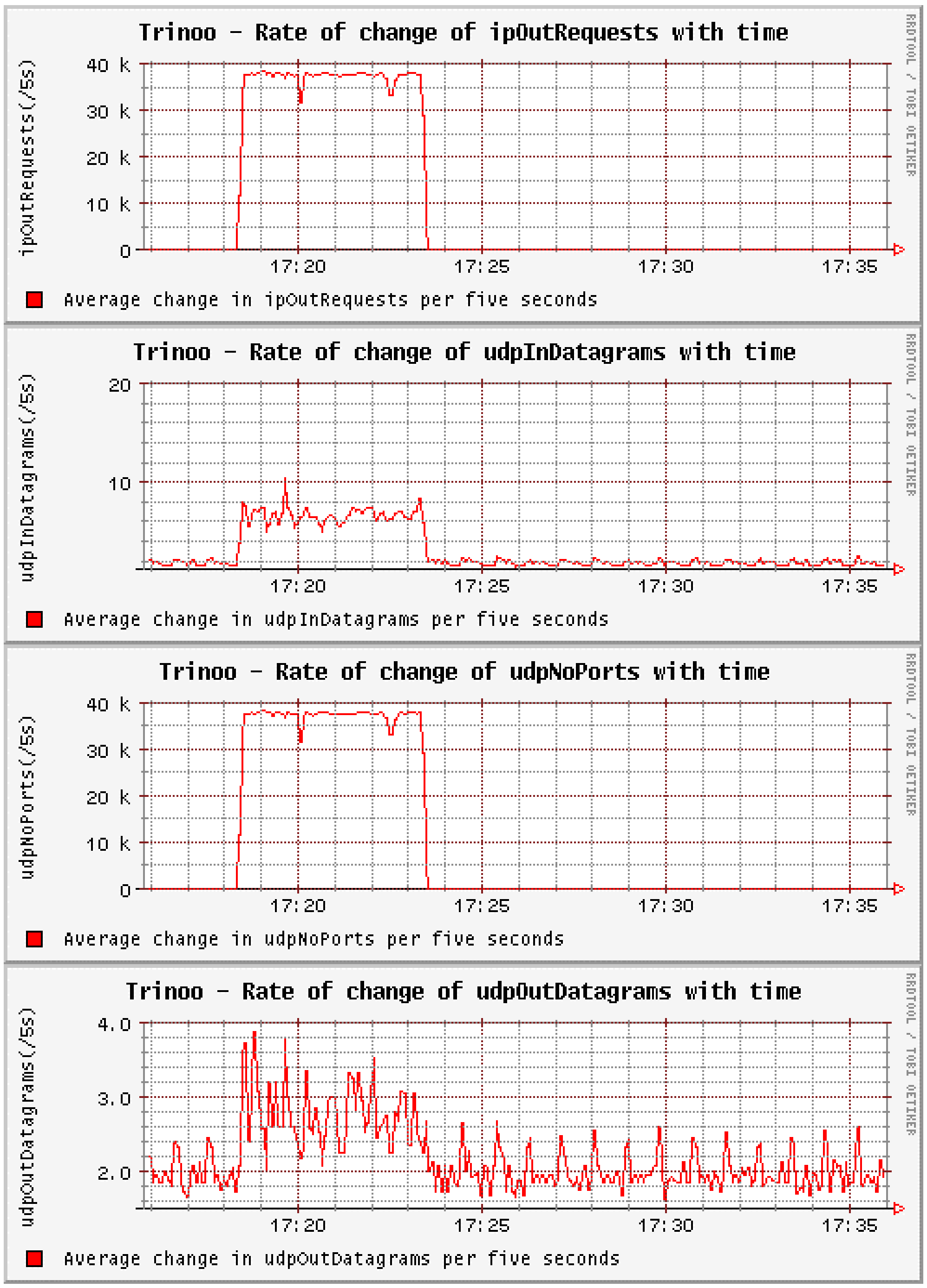
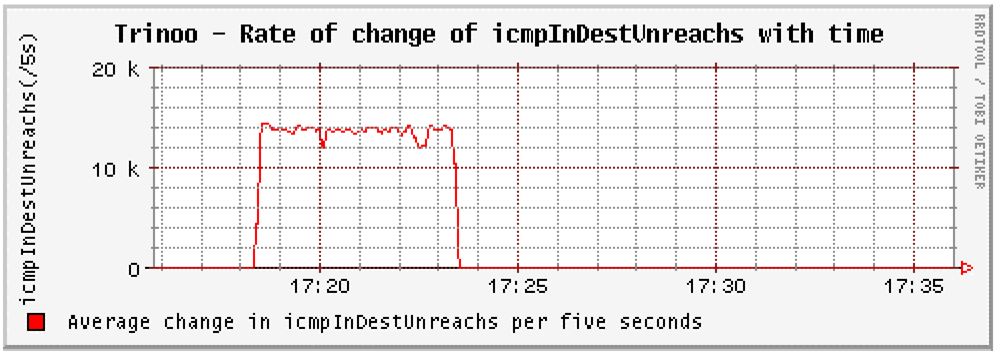
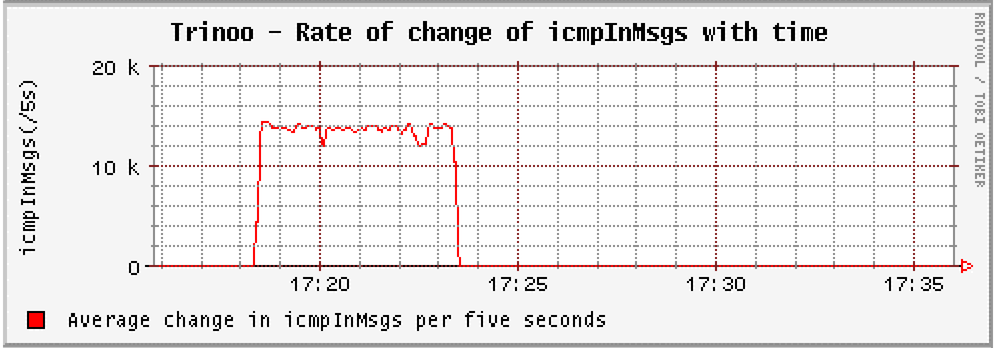


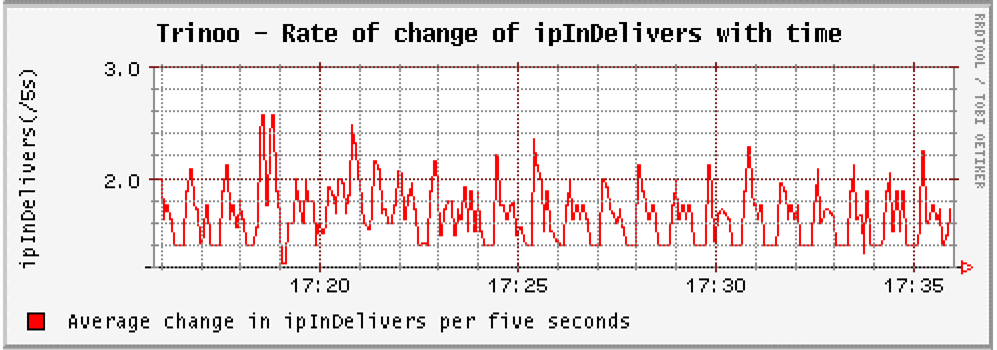
Figure 6. Trinoo Attack - Changes In Target MIB Variables.

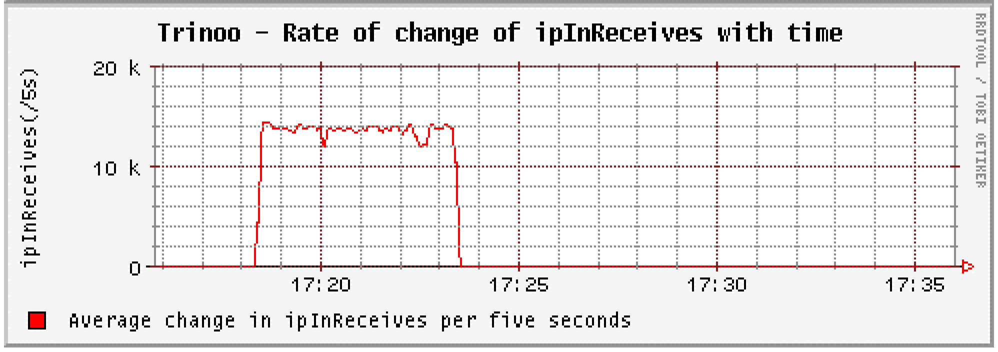
2. Changes in the Attacker1 MIB Variables

Figure 7 shows the average change in the attacker1 MIB variables per five-second interval in case of a trinoo attack.









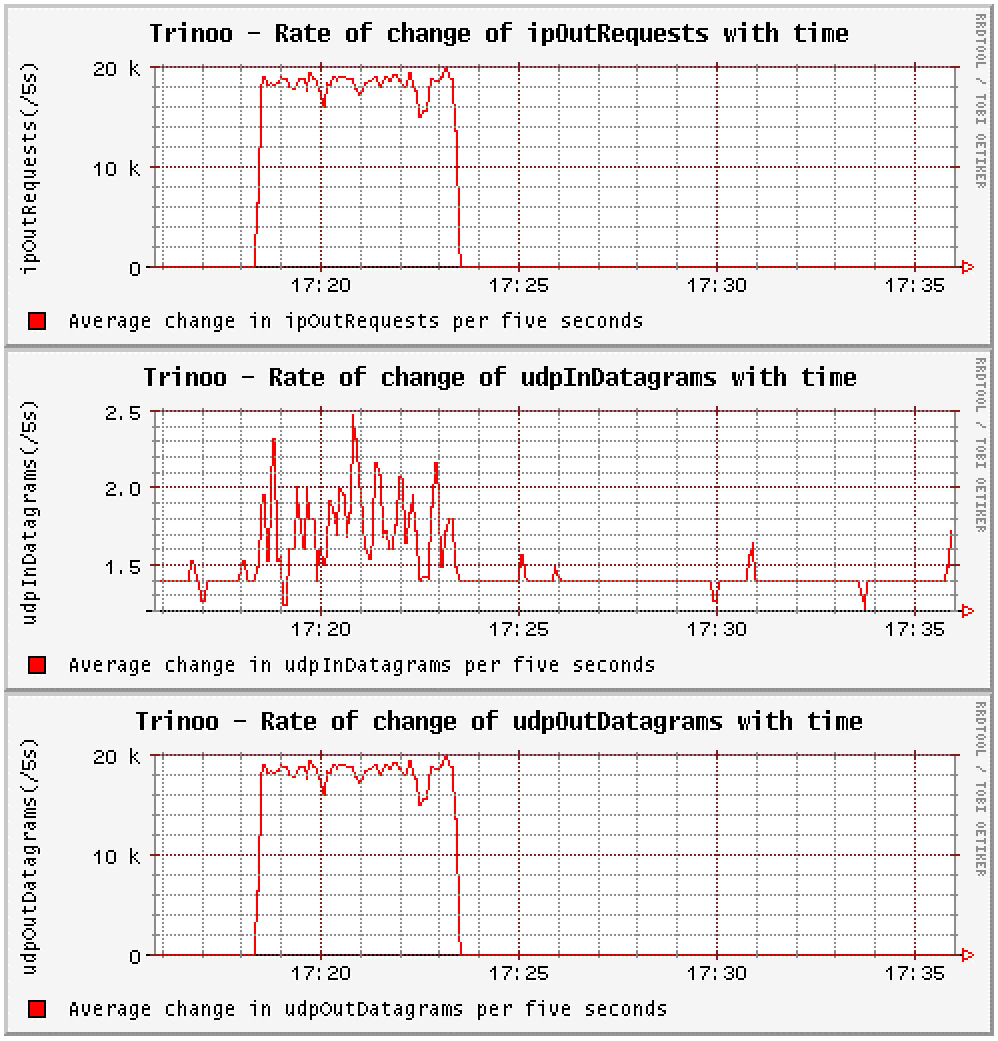
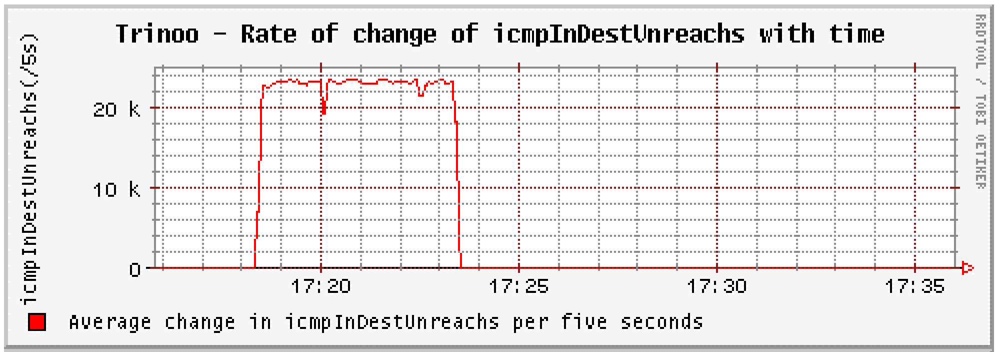
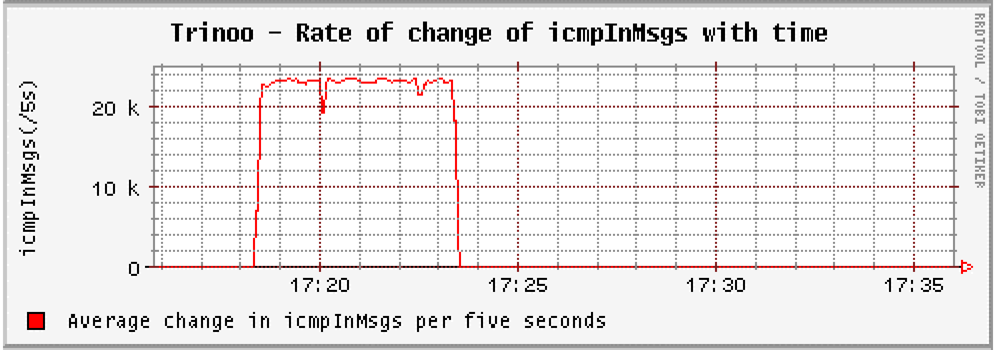


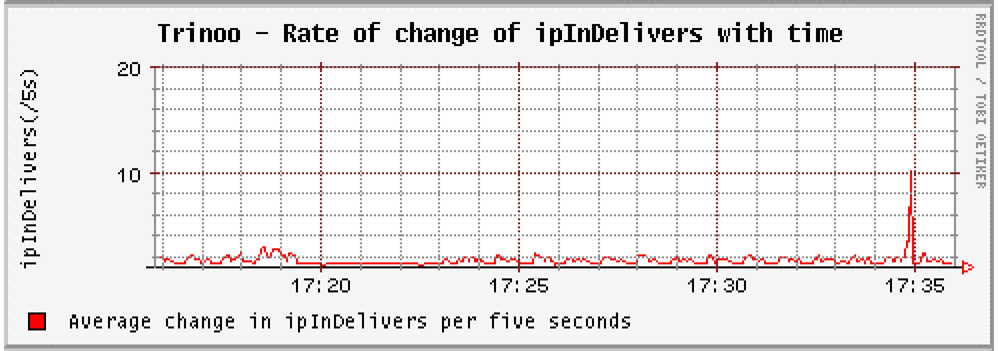
Figure 7. Trinoo Attack - Changes In Attacker1 MIB Variables.

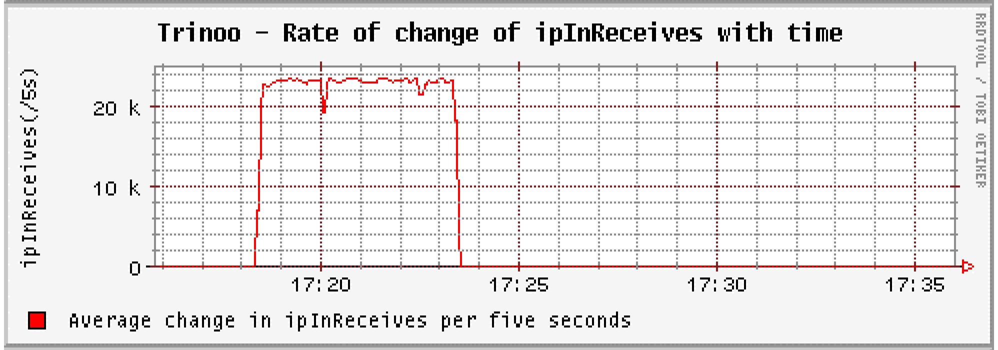
3. Changes in the Attacker2 MIB Variables

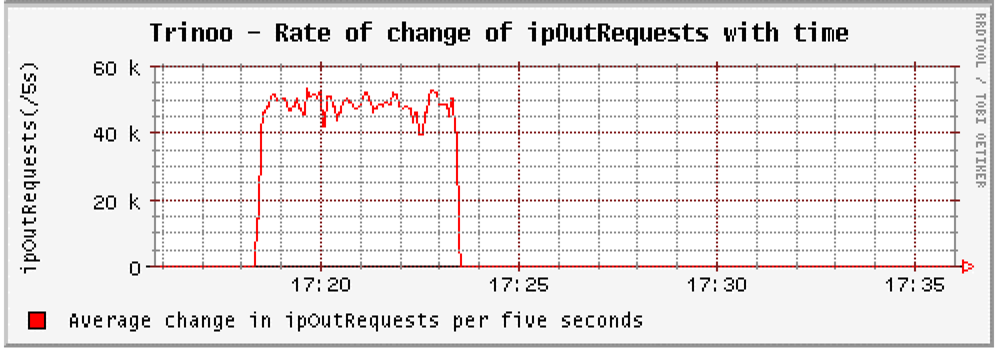
Figure 8 shows the average change in the attacker2 MIB variables per five-second interval in case of a trinoo attack.


Trinoo - Rate of change of icmpInDestUnreachs with time
icmpInDestUnreachs(/5s)
20 k
10 k
0
17:20
17:25
17:30
17:35
🟥 Average change in icmpInDestUnreachs per five seconds
RRDTOOL / TOBI OETIKER


Trinoo - Rate of change of icmpInMsgs with time
icmpInMsgs(/5s)
20 k
10 k
0
17:20
17:25
17:30
17:35
🟥 Average change in icmpInMsgs per five seconds
RRDTOOL / TOBI OETIKER


Trinoo - Rate of change of ipInDelivers with time
ipInDelivers(/5s)
20
10
17:20
17:25
17:30
17:35
🟥 Average change in ipInDelivers per five seconds
RRDTOOL / TOBI OETIKER


Trinoo - Rate of change of ipInReceives with time
ipInReceives(/5s)
20 k
10 k
0
17:20
17:25
17:30
17:35
🟥 Average change in ipInReceives per five seconds
RRDTOOL / TOBI OETIKER


Trinoo - Rate of change of ipOutRequests with time
ipOutRequests(/5s)
60 k
40 k
20 k
0
17:20
17:25
17:30
17:35
🟥 Average change in ipOutRequests per five seconds
RRDTOOL / TOBI OETIKER

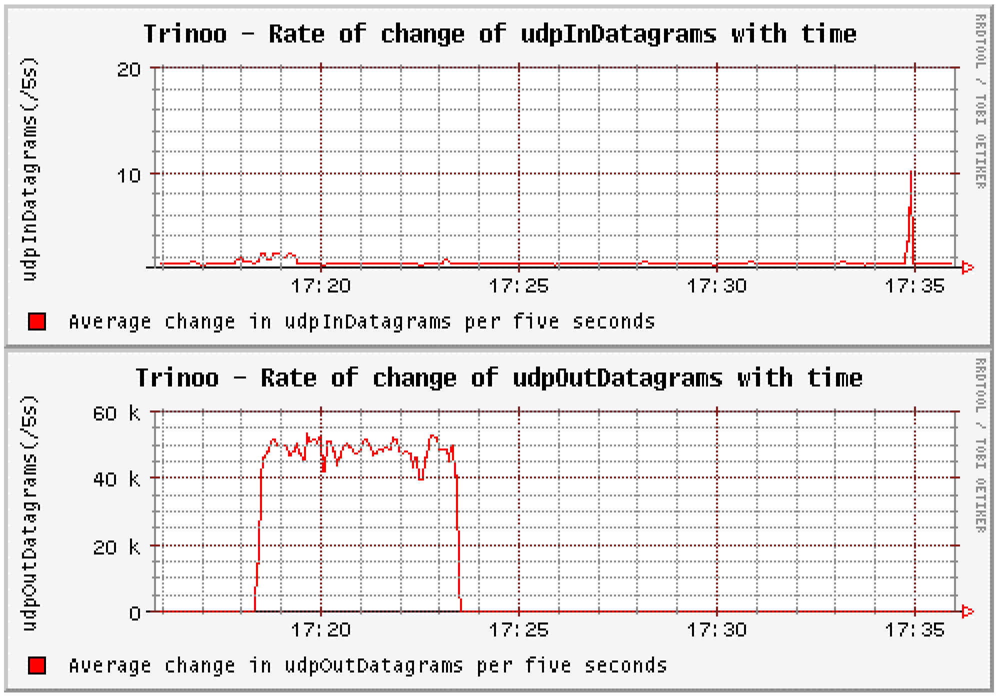


Figure 8. Trinoo Attack - Changes In Attacker2 MIB Variables.

B. TFN

1. Ping (ICMP) Flood

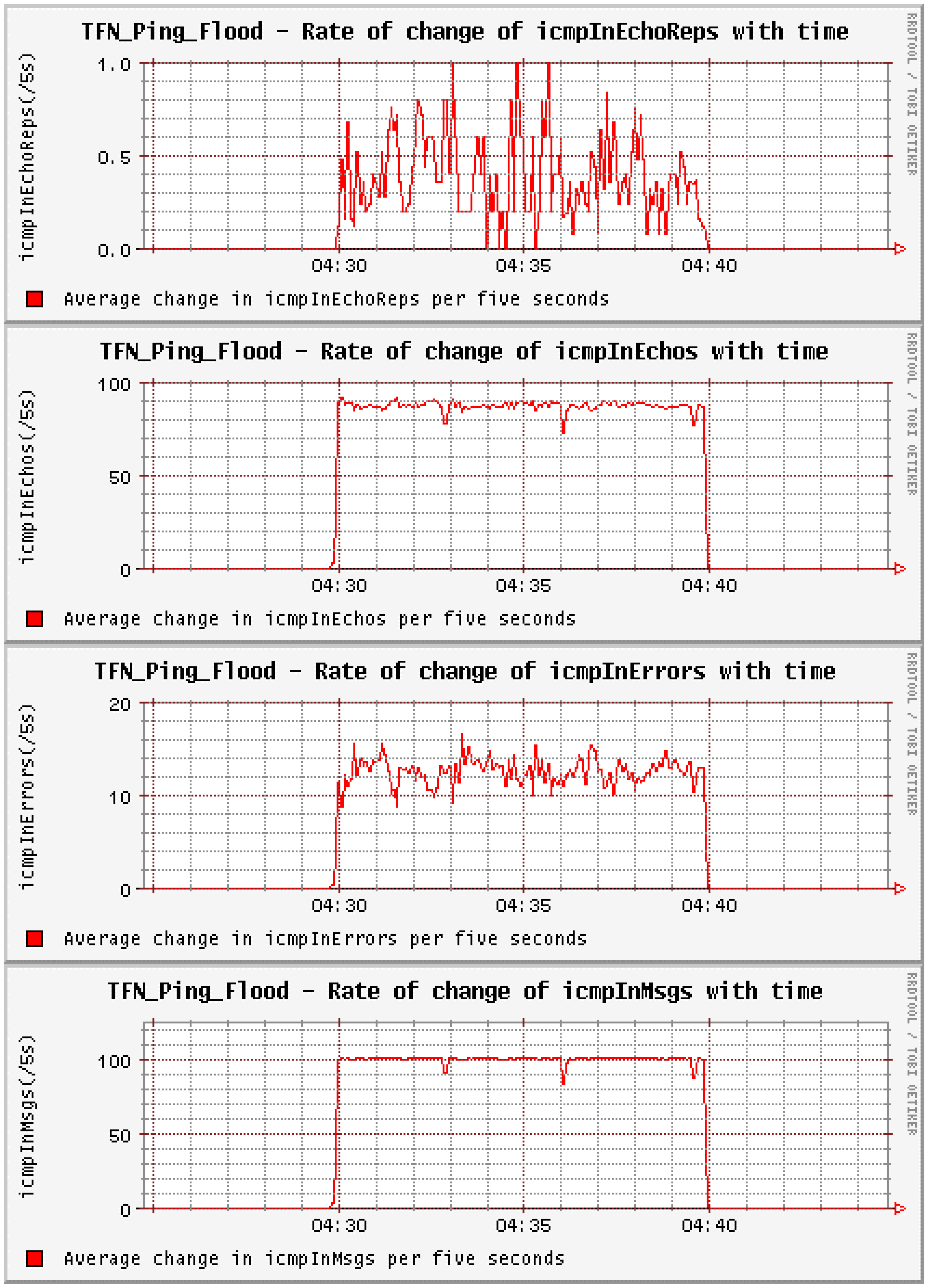
The affected variables at the target and at the attacker in a TFN ping-flood attack are shown in table 2.

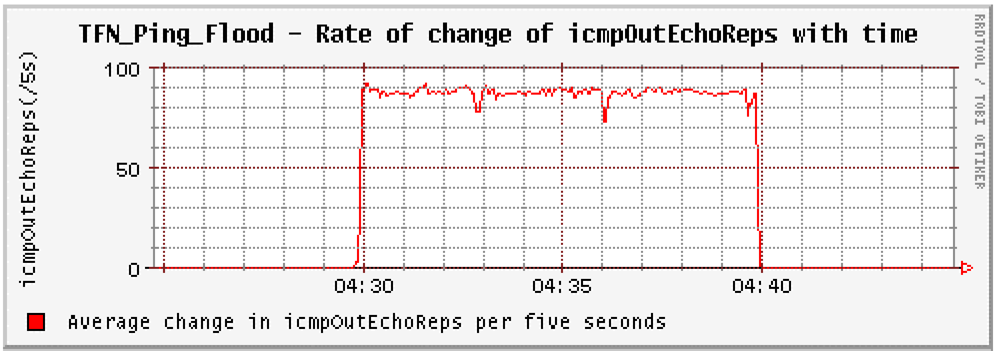
Sl. No.	Target	Attacker
1.	icmp.icmpInEchoReps.0	icmp.icmpInEchoReps.0
2.	**icmp.icmpInEchos.0**	icmp.icmpInMsgs.0
3.	icmp.icmpInErrors.0	**ip.ipOutRequests.0**
4.	**icmp.icmpInMsgs.0**	
5.	**icmp.icmpOutEchoReps.0**	
6.	**icmp.icmpOutMsgs.0**	
7.	**ip.ipInDelivers.0**	
8.	**ip.ipInReceives.0**	
9.	**ip.ipOutRequests.0**	
10.	udp.udpNoPorts.0	

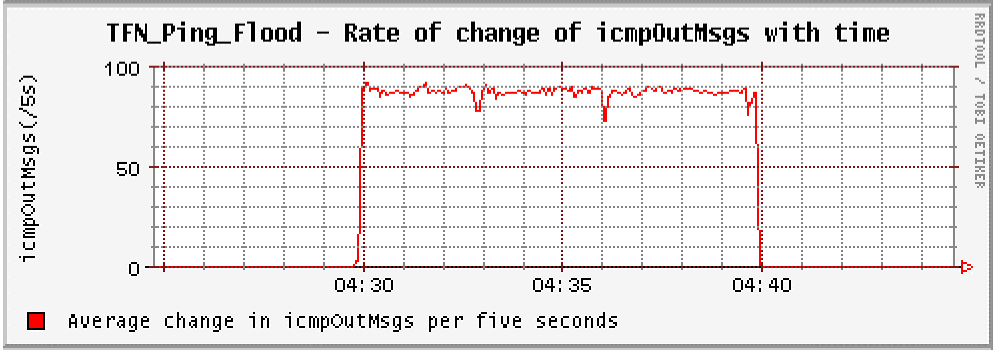
Table 2. Affected MIB Variables In TFN Ping Flood Attack.

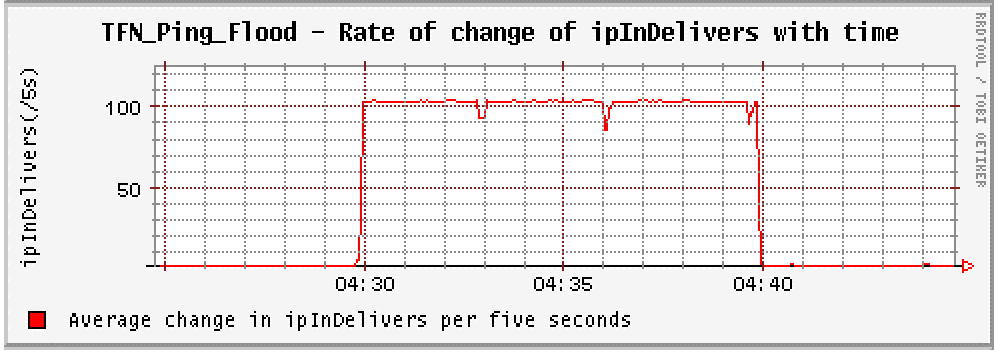
a. Changes in the Target MIB Variables

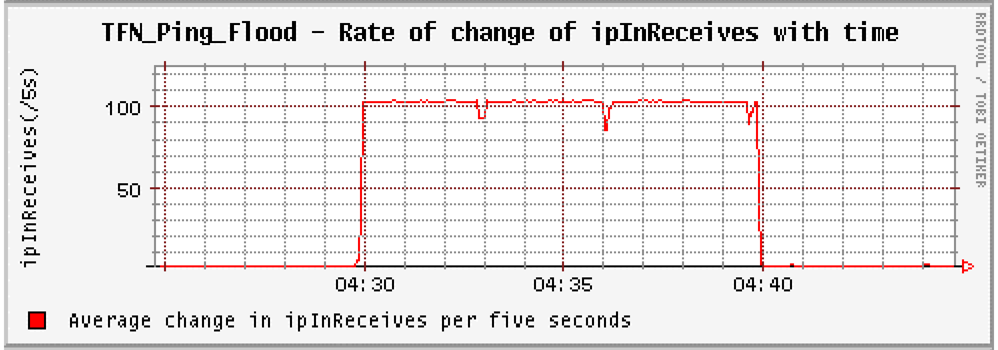
Figure 9 below shows the average change in the target MIB variables per five-second interval for a TFN ping-flood attack.

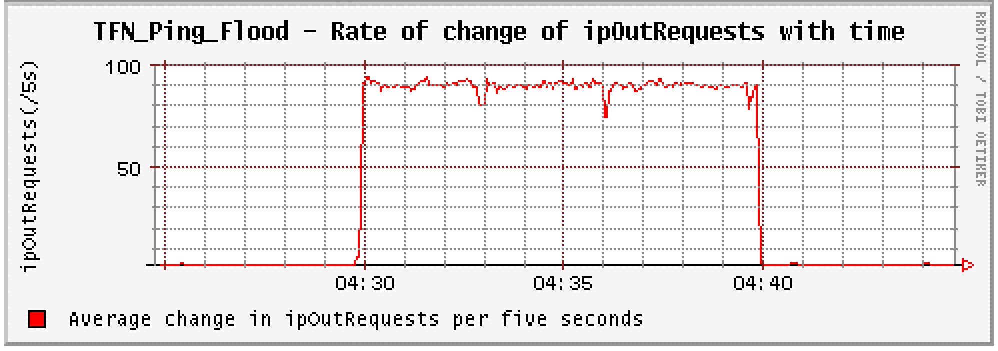



TFN_Ping_Flood - Rate of change of icmpOutEchoReps with time
icmpOutEchoReps(/5s)
100
50
0
04:30
04:35
04:40
🟥 Average change in icmpOutEchoReps per five seconds
RRDTOOL / TOBI OETIKER


TFN_Ping_Flood - Rate of change of icmpOutMsgs with time
icmpOutMsgs(/5s)
100
50
0
04:30
04:35
04:40
🟥 Average change in icmpOutMsgs per five seconds
RRDTOOL / TOBI OETIKER


TFN_Ping_Flood - Rate of change of ipInDelivers with time
ipInDelivers(/5s)
100
50
04:30
04:35
04:40
🟥 Average change in ipInDelivers per five seconds
RRDTOOL / TOBI OETIKER


TFN_Ping_Flood - Rate of change of ipInReceives with time
ipInReceives(/5s)
100
50
04:30
04:35
04:40
🟥 Average change in ipInReceives per five seconds
RRDTOOL / TOBI OETIKER


TFN_Ping_Flood - Rate of change of ipOutRequests with time
ipOutRequests(/5s)
100
50
04:30
04:35
04:40
🟥 Average change in ipOutRequests per five seconds
RRDTOOL / TOBI OETIKER

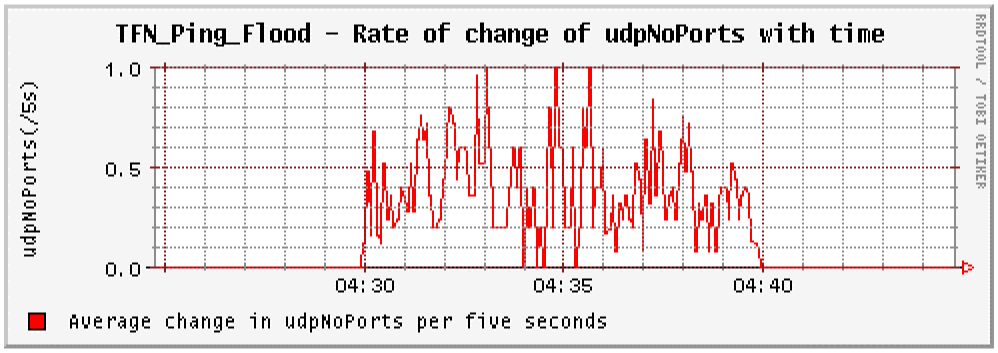
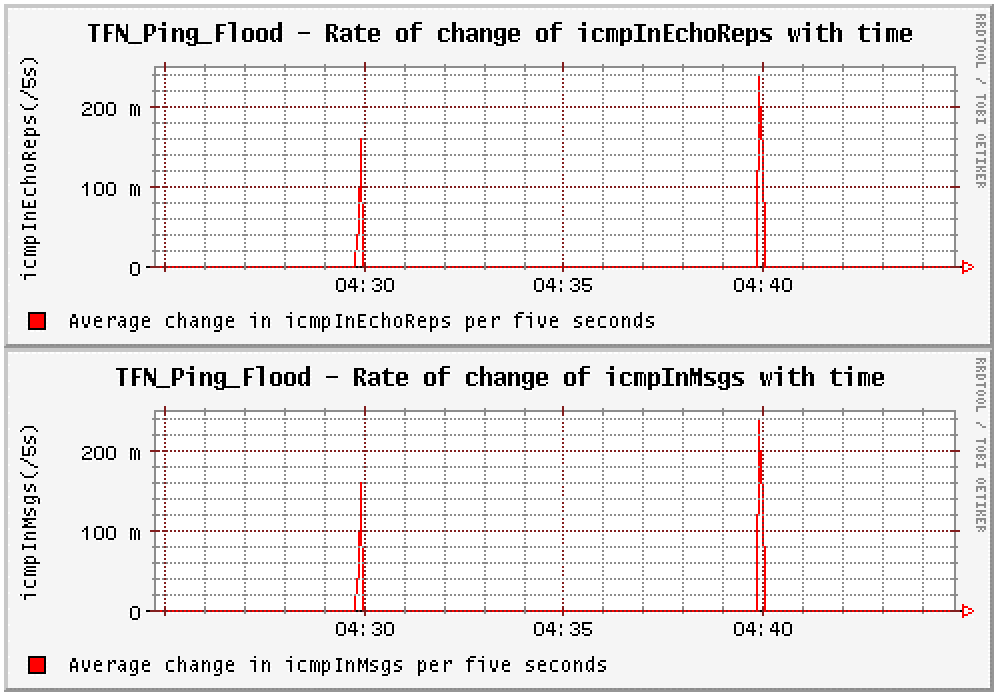


Figure 9. TFN Ping Flood Attack - Changes In Target MIB Variables.

b. Changes in the Attacker-1 MIB Variables

Figure 10 shows the average changes in the attacker1 MIB variables per five-second interval when under a TFN ping-flood attack.



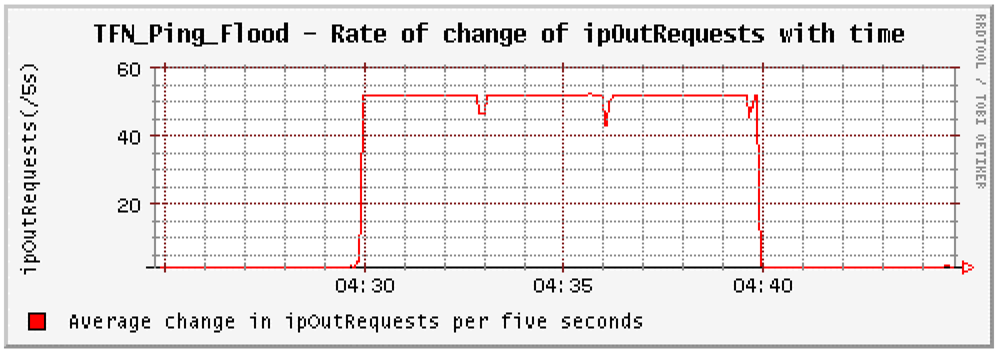
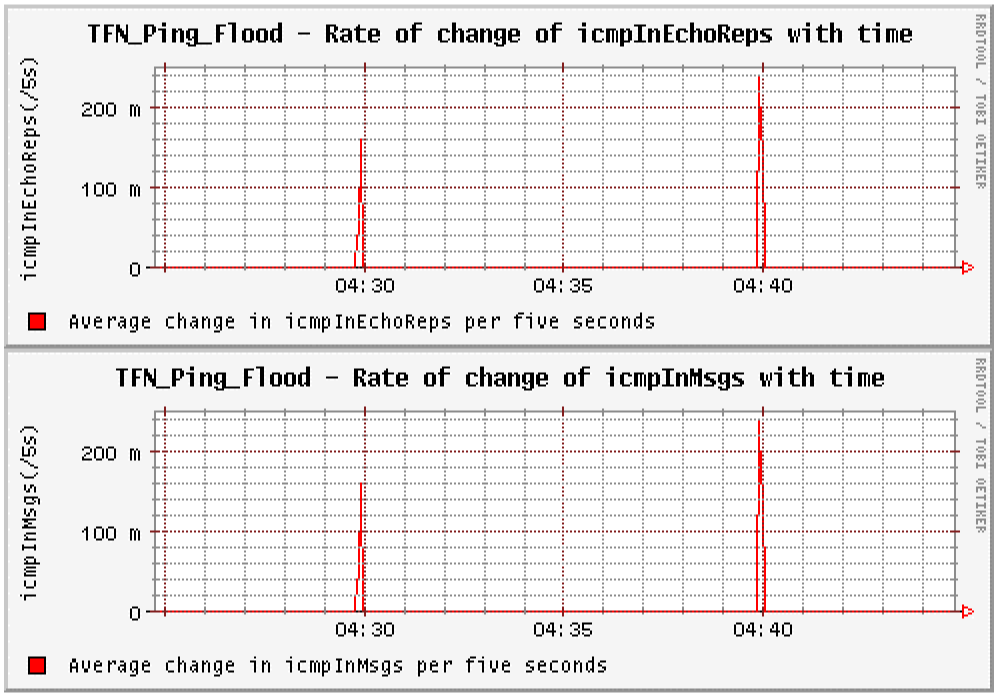


Figure 10. TFN Ping Flood Attack - Changes In Attacker1 MIB Variables.

c. Changes in the Attacker-2 MIB Variables

Figure 11 shows the average changes in the attacker2 MIB variables per five-second interval when under a TFN ping-flood attack.



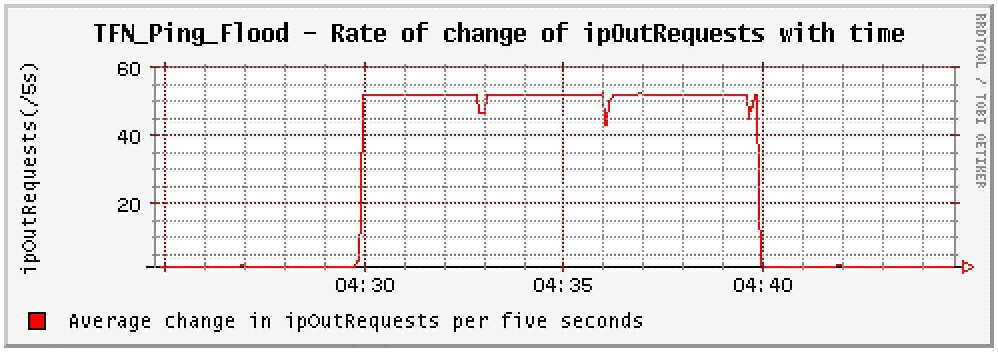


Figure 11. TFN Ping Flood Attack - Changes In Attacker2 MIB Variables.

2. SYN Flood

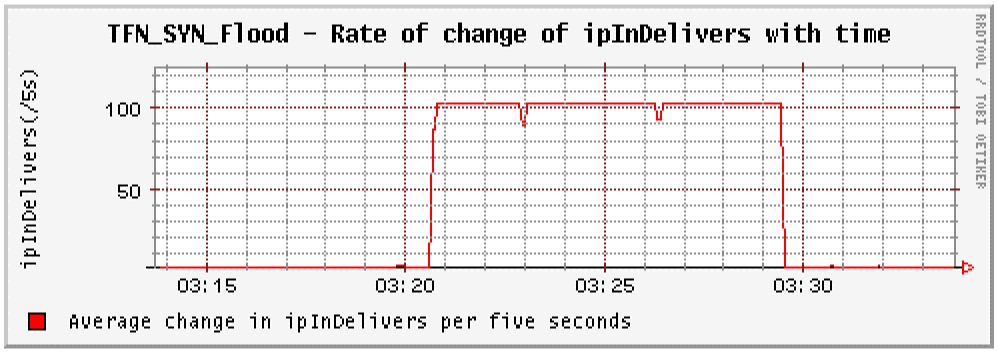
The affected variables at the target and at the attacker in a TFN syn-flood attack are shown in table 3.

Sl. No.	Target	Attacker
1.	**ip.ipInDelivers.0**	icmp.icmpInEchoReps.0
2.	**ip.ipInReceives.0**	icmp.icmpInMsgs.0
3.	**tcp.tcpInErrs.0**	**ip.ipOutRequests.0**
4.	**tcp.tcpInSegs.0**	

Table 3. Affected MIB Variables In TFN Syn Flood Attack.

a. Changes in the Target MIB Variables

Figure 12 below shows the average change in the target MIB variables per five-second interval for a TFN syn-flood attack.



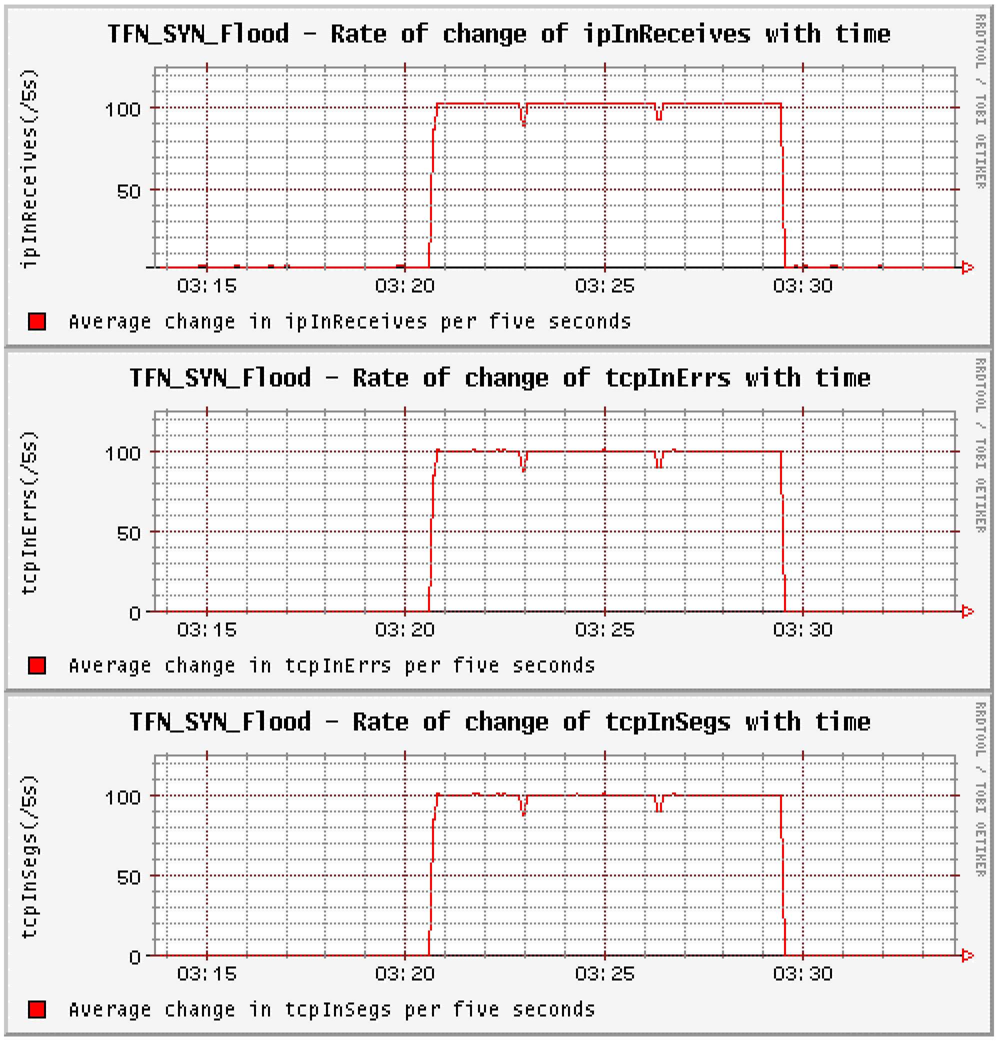


Figure 12. TFN Syn Flood Attack - Changes In Target MIB Variables.

b. Changes in the Attacker-1 MIB Variables

Figure 13 shows the average changes in the attacker1 MIB variables per five-second interval when under a TFN syn-flood attack.

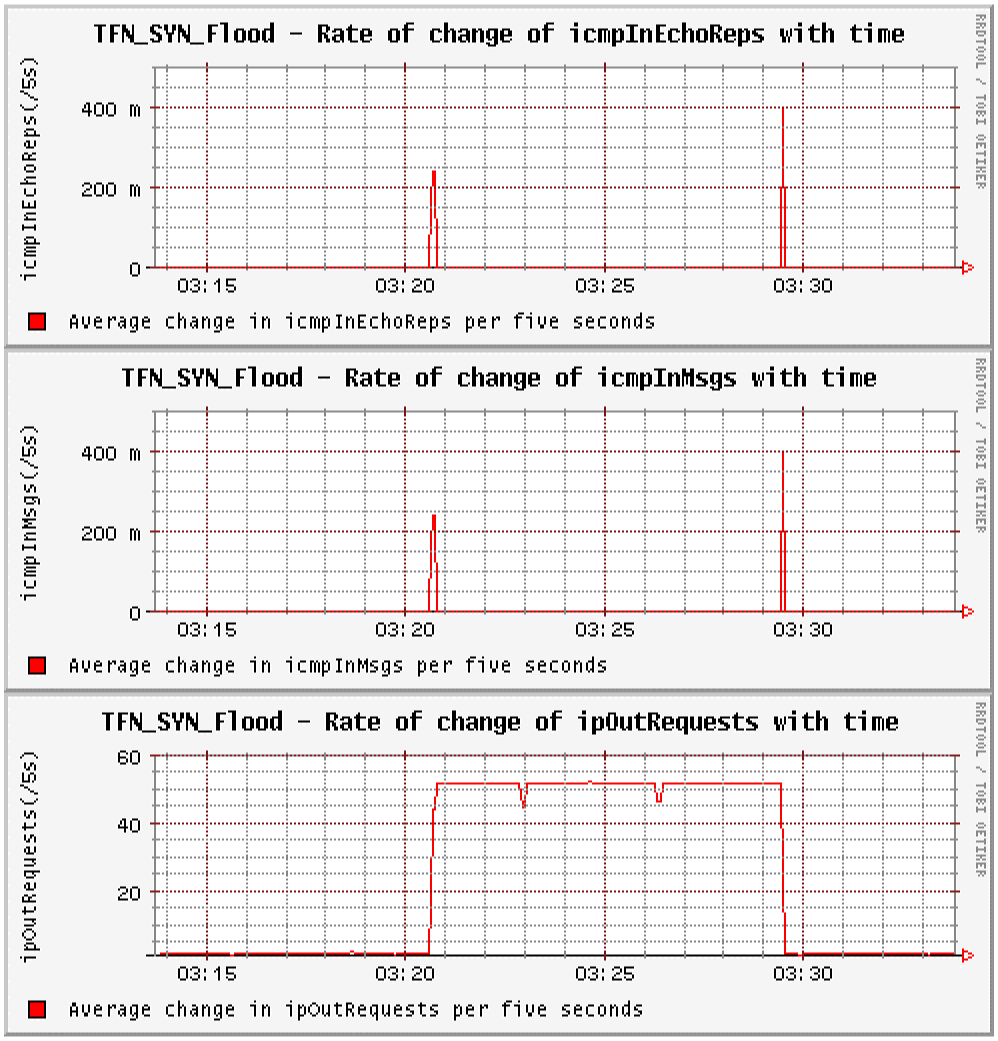


Figure 13. TFN Syn Flood Attack - Changes In Attacker1 MIB Variables.

c. Changes in the Attacker-2 MIB Variables

Figure 14 shows the average changes in the attacker2 MIB variables per five-second interval when under a TFN syn-flood attack.

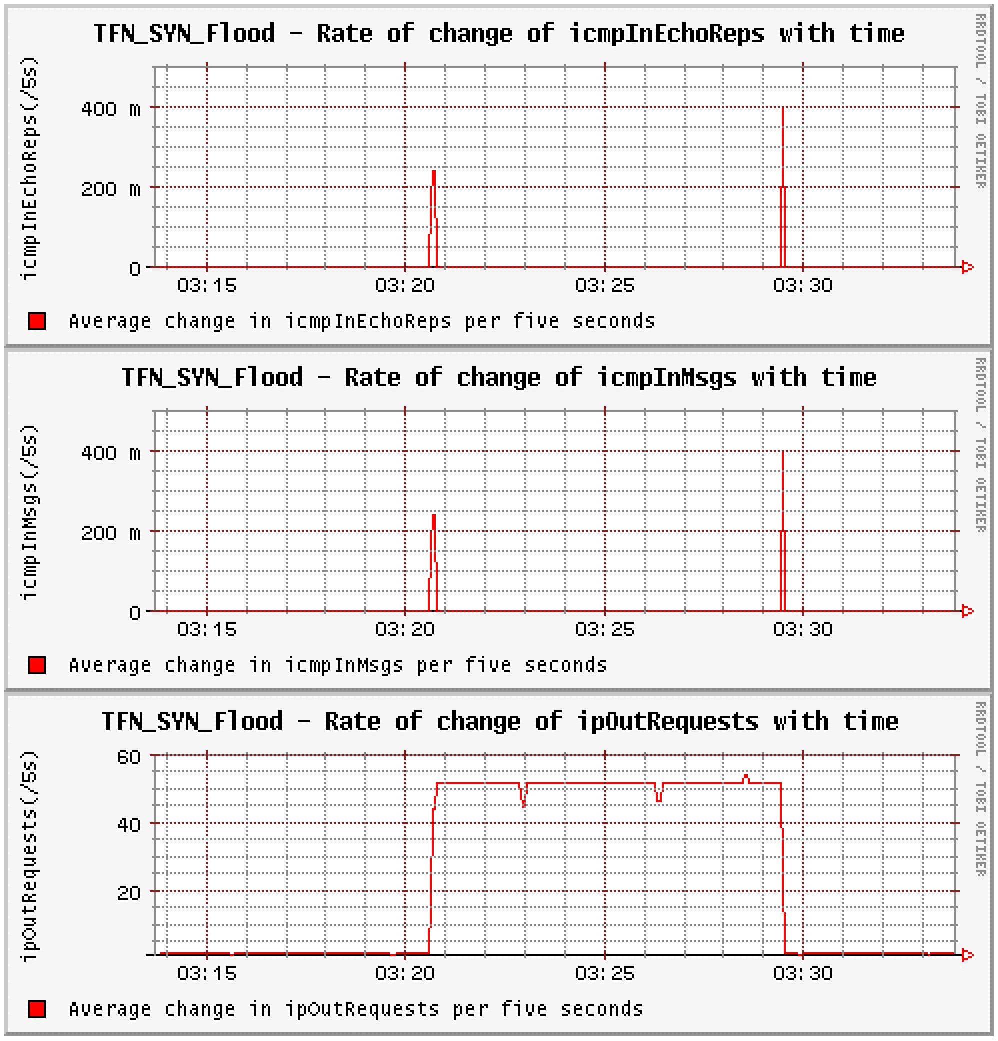


Figure 14. TFN Syn Flood Attack - Changes In Attacker2 MIB Variables.

3. UDP Flood

The affected variables at the target and at the attacker in a TFN UDP-Flood attack are shown in table 4.

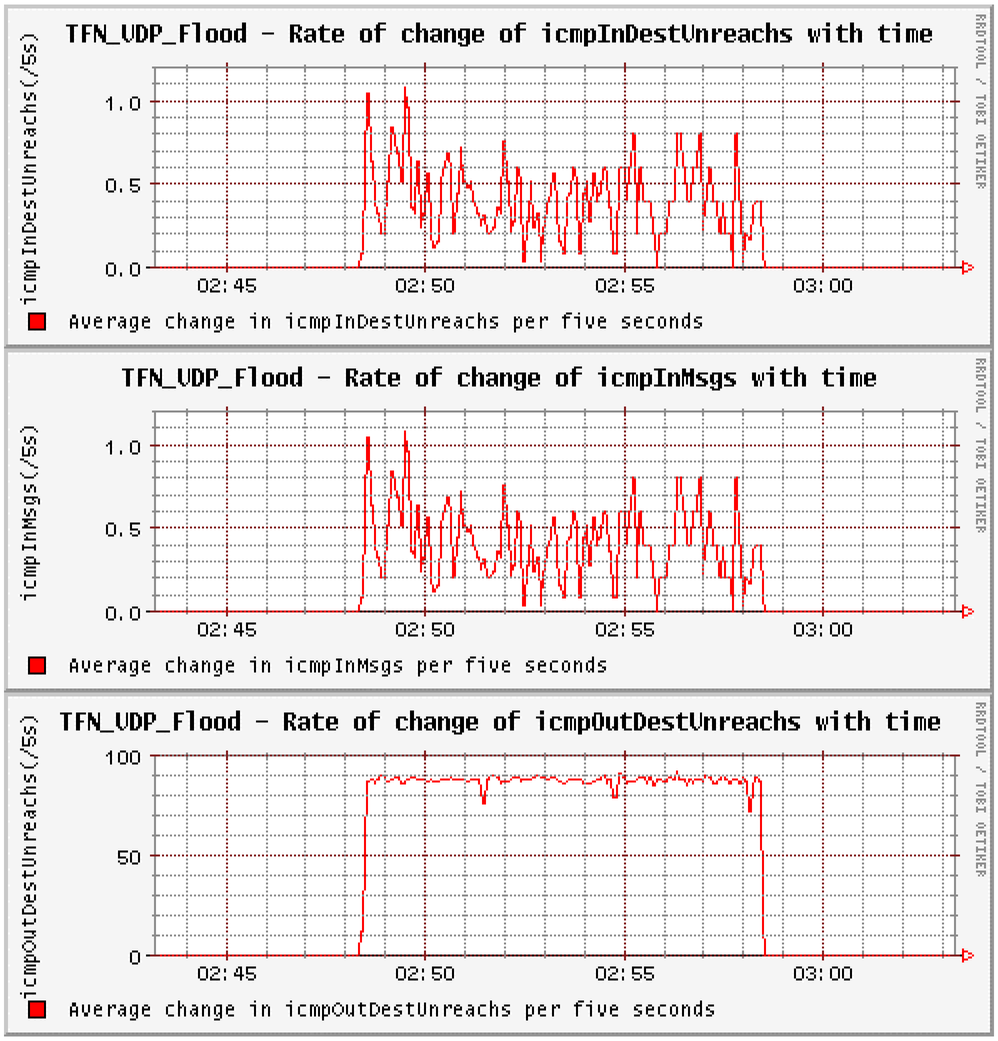
Sl. No.	Target	Attacker
1.	**icmp.icmpInDestUnreachs.0**	icmp.icmpInEchoReps.0
2.	**icmp.icmpInMsgs.0**	icmp.icmpInMsgs.0
3.	**icmp.icmpOutDestUnreachs.0**	**ip.ipOutRequests.0**
4.	**icmp.icmpOutMsgs.0**	

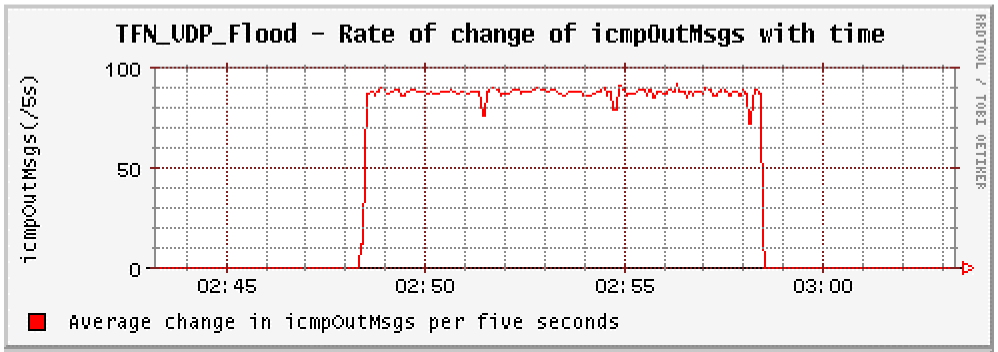
5.	**ip.ipInDelivers.0**	
6.	**ip.ipInReceives.0**	
7.	**ip.ipOutRequests.0**	
8.	**udp.udpInErrors.0**	
9.	**udp.udpNoPorts.0**	

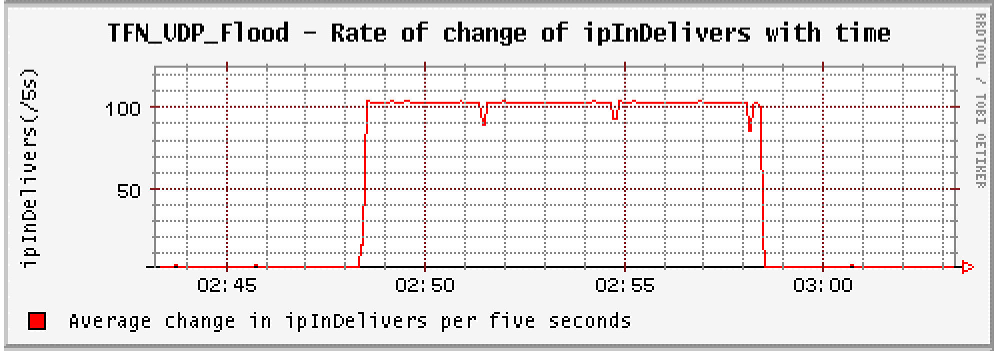
Table 4. Affected MIB Variables In TFN UDP Flood Attack.

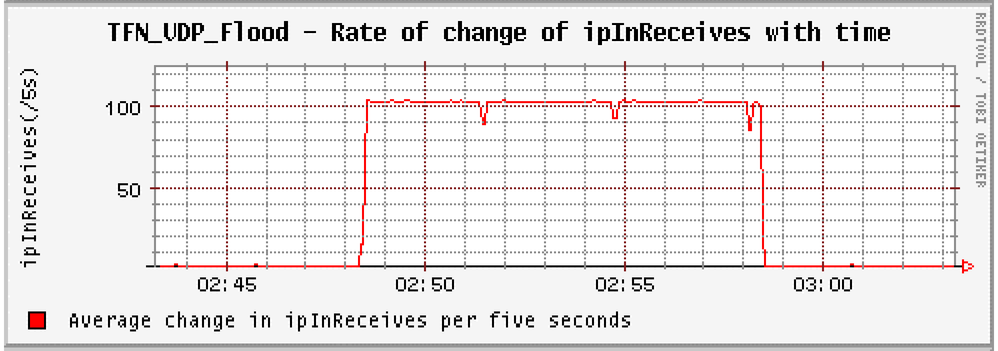
a. Changes in the Target MIB variables

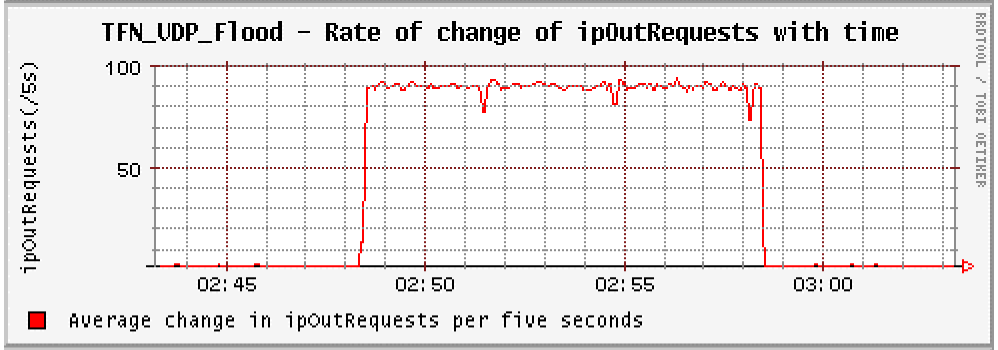
Figure 15 below shows the average change in the target MIB variables per five-second interval for a TFN UDP-flood attack.

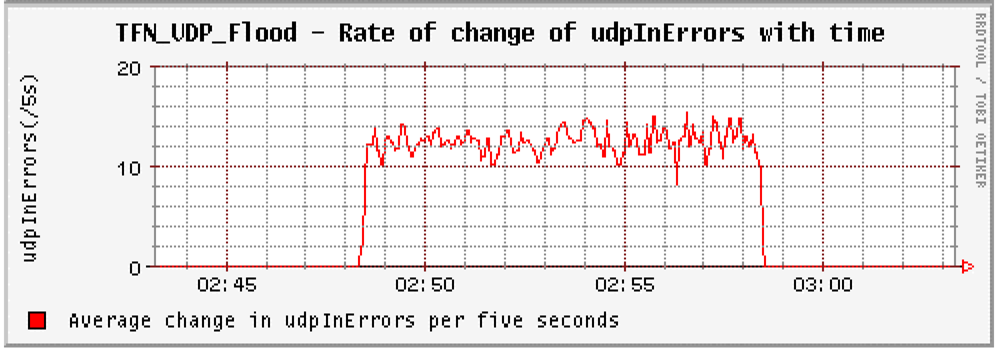



TFN_UDP_Flood - Rate of change of icmpOutMsgs with time
icmpOutMsgs(/5s)
100
50
0
02:45
02:50
02:55
03:00
🟥 Average change in icmpOutMsgs per five seconds
RRDTOOL / TOBI OETIKER


TFN_UDP_Flood - Rate of change of ipInDelivers with time
ipInDelivers(/5s)
100
50
02:45
02:50
02:55
03:00
🟥 Average change in ipInDelivers per five seconds
RRDTOOL / TOBI OETIKER


TFN_UDP_Flood - Rate of change of ipInReceives with time
ipInReceives(/5s)
100
50
02:45
02:50
02:55
03:00
🟥 Average change in ipInReceives per five seconds
RRDTOOL / TOBI OETIKER


TFN_UDP_Flood - Rate of change of ipOutRequests with time
ipOutRequests(/5s)
100
50
02:45
02:50
02:55
03:00
🟥 Average change in ipOutRequests per five seconds
RRDTOOL / TOBI OETIKER


TFN_UDP_Flood - Rate of change of udpInErrors with time
udpInErrors(/5s)
20
10
0
02:45
02:50
02:55
03:00
🟥 Average change in udpInErrors per five seconds
RRDTOOL / TOBI OETIKER

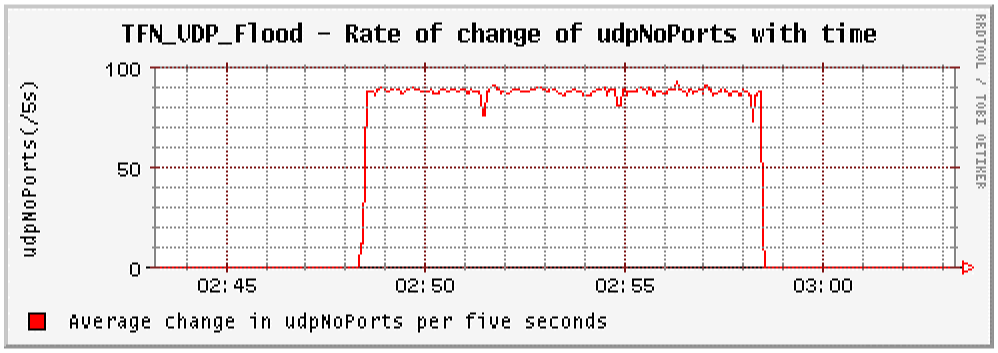
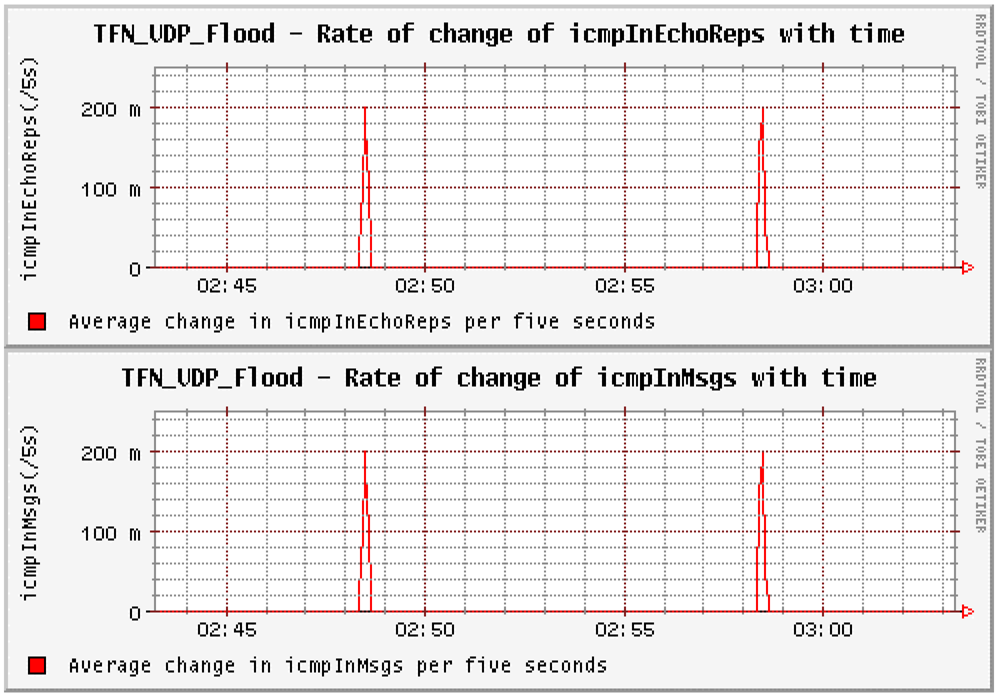


Figure 15. TFN UDP Flood Attack - Changes In Target MIB Variables.

b. Changes in the Attacker-1 MIB Variables

Figure 16 shows the average changes in the attacker1 MIB variables per five-second interval when under a TFN UDP-flood attack.



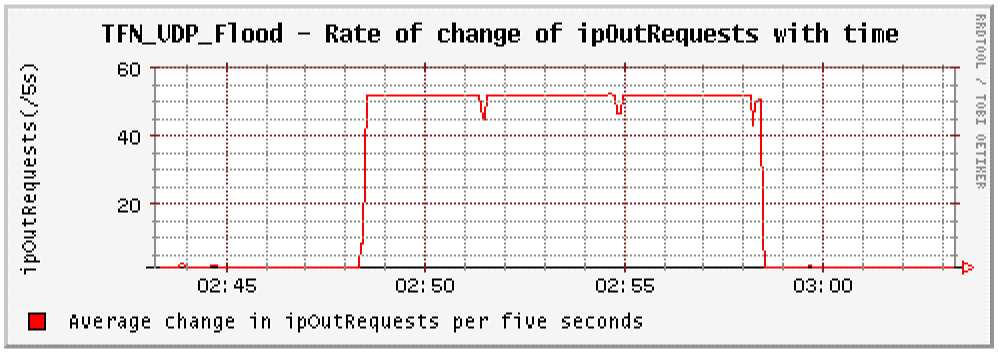
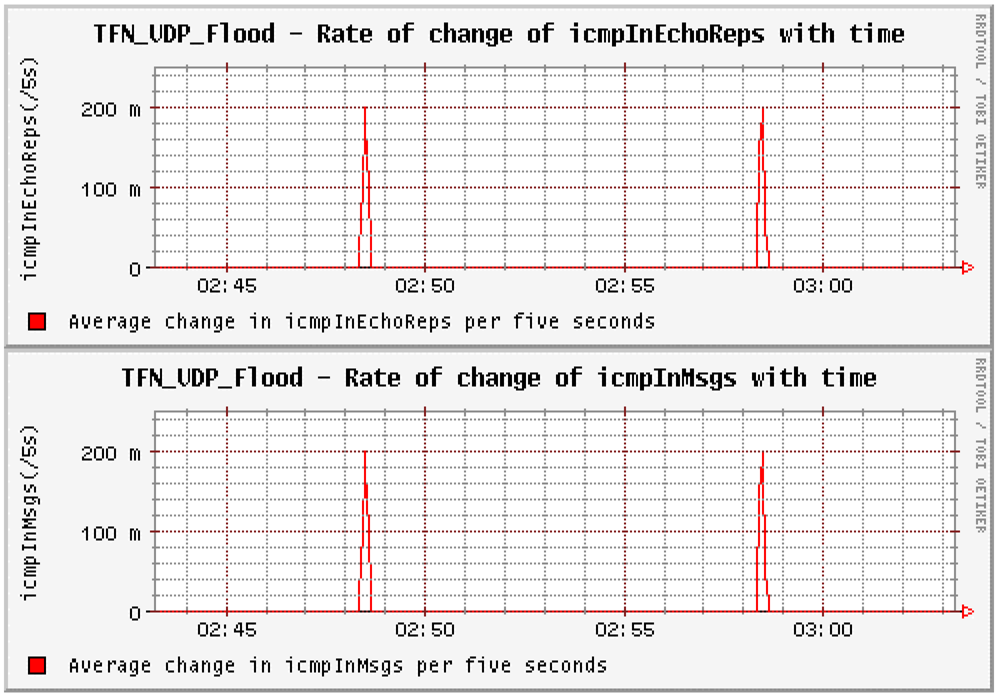


Figure 16. TFN UDP Flood Attack - Changes In Attacker1 MIB Variables.

c. Changes in the Attacker-2 MIB Variables

Figure 17 shows the average changes in the attacker2 MIB variables per five-second interval when under a TFN UDP-flood attack.



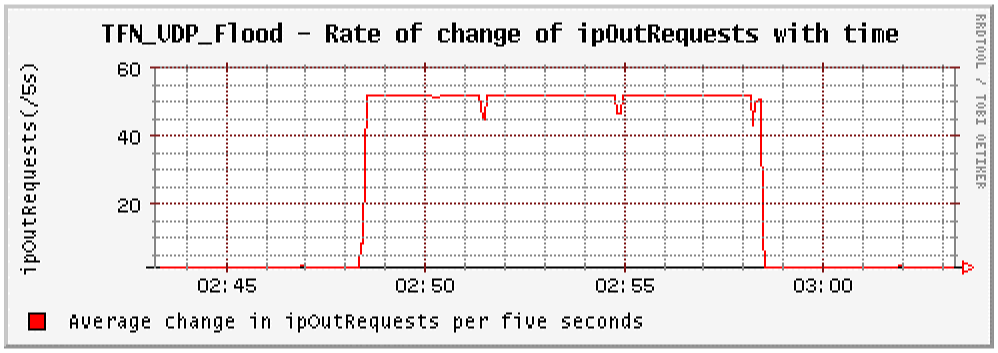


Figure 17. TFN UDP Flood Attack - Changes In Attacker2 MIB Variables.

C. TFN2K

1. Ping Flood

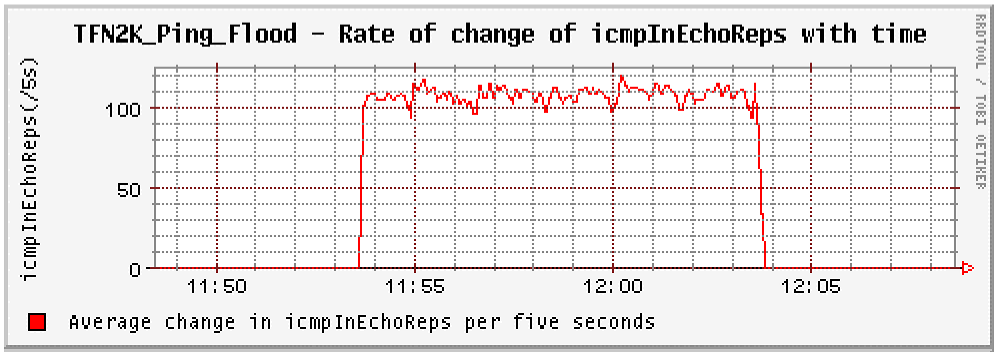
The affected variables at the target and at the attacker in a TFN2K ping-flood attack are shown in table 5.

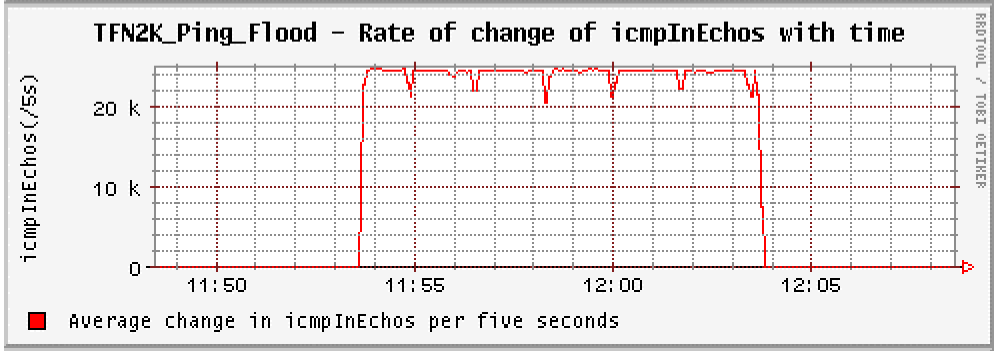
Sl. No.	Target	Attacker
1.	icmp.icmpInEchoReps.0	icmp.icmpInEchoReps.0
2.	**icmp.icmpInEchos.0**	icmp.icmpInMsgs.0
3.	**icmp.icmpInErrors.0**	ip.ipInDelivers.0
4.	**icmp.icmpInMsgs.0**	ip.ipInReceives.0
5.	**icmp.icmpOutEchoReps.0**	**ip.ipOutRequests.0**
6.	**icmp.icmpOutMsgs.0**	tcp.tcpInSegs.0
7.	**ip.ipInDelivers.0**	udp.udpInErrors.0
8.	**ip.ipInReceives.0**	
9.	**ip.ipOutRequests.0**	
10.	udp.udpNoPorts.0	

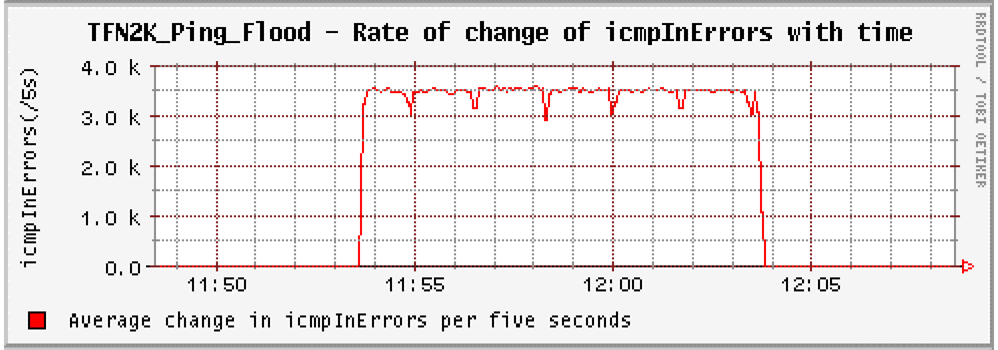
Table 5. Affected MIB Variables In TFN2K Ping Flood Attack.

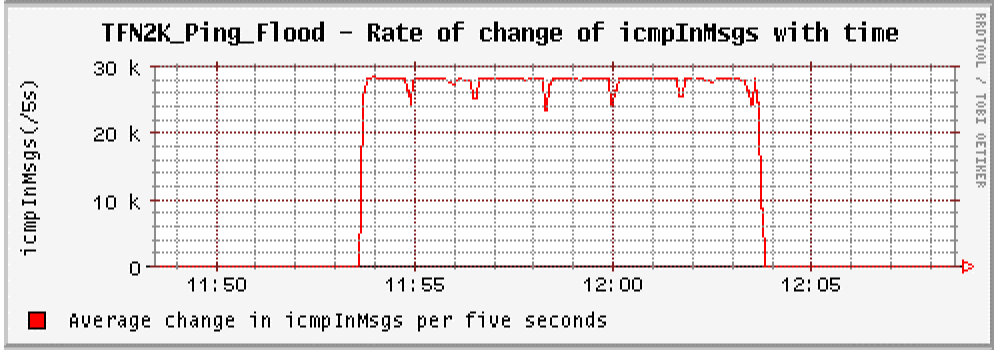
a. Changes in the Target MIB Variables

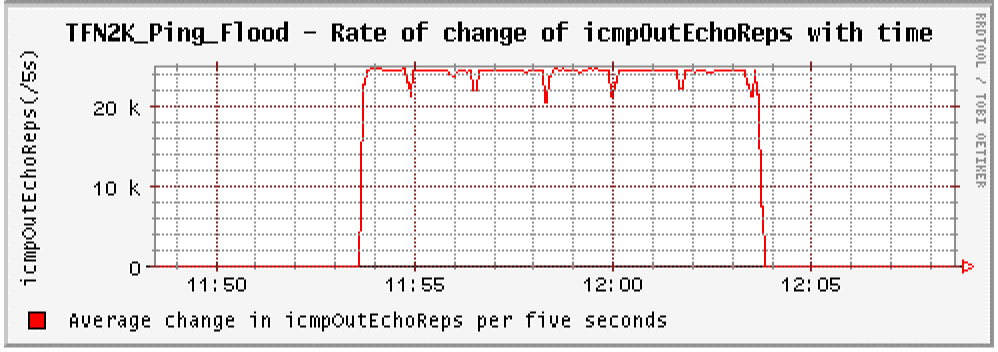
Figure 18 below shows the average change in the target MIB variables per five-second interval for a TFN2K ping-flood attack.

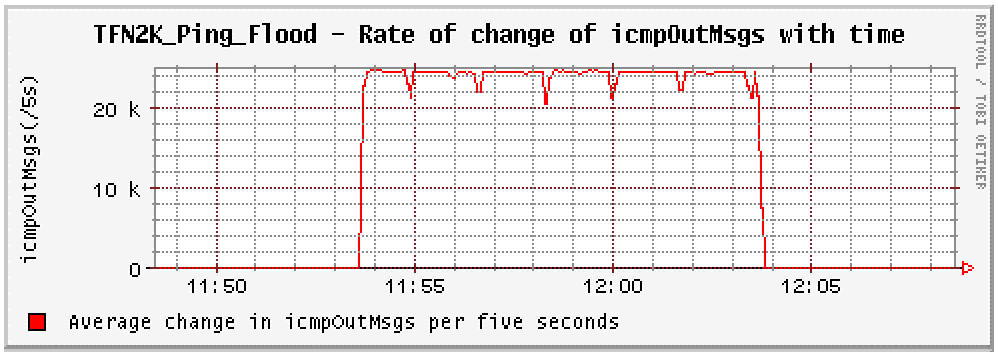

TFN2K_Ping_Flood - Rate of change of icmpInEchoReps with time
icmpInEchoReps(/5s)
100
50
0
11:50
11:55
12:00
12:05
🟥 Average change in icmpInEchoReps per five seconds
RRDTOOL / TOBI OETIKER

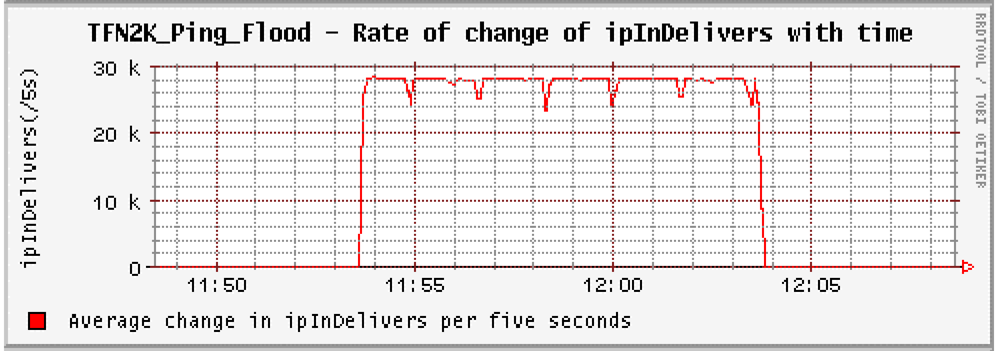

TFN2K_Ping_Flood - Rate of change of icmpInEchos with time
icmpInEchos(/5s)
20 k
10 k
0
11:50
11:55
12:00
12:05
🟥 Average change in icmpInEchos per five seconds
RRDTOOL / TOBI OETIKER

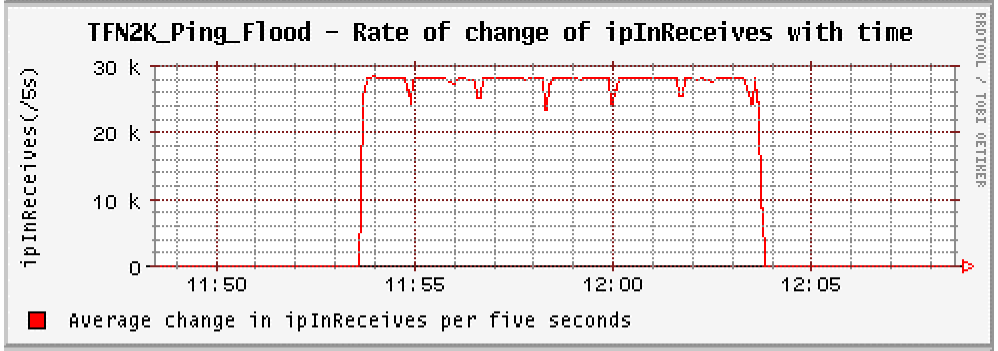

TFN2K_Ping_Flood - Rate of change of icmpInErrors with time
icmpInErrors(/5s)
4.0 k
3.0 k
2.0 k
1.0 k
0.0
11:50
11:55
12:00
12:05
🟥 Average change in icmpInErrors per five seconds
RRDTOOL / TOBI OETIKER

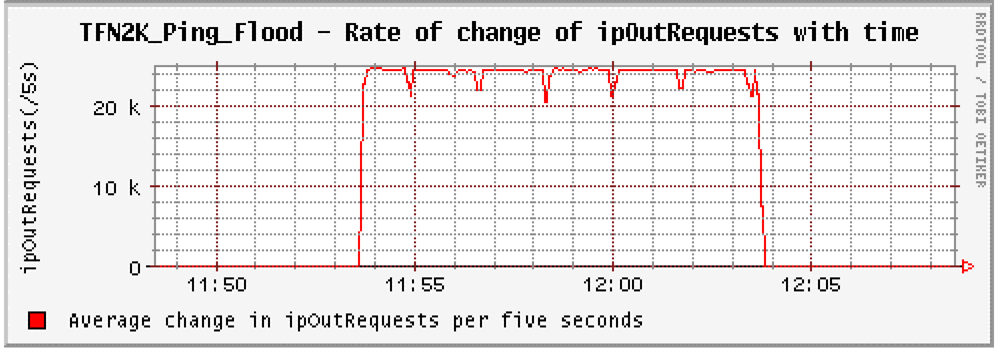

TFN2K_Ping_Flood - Rate of change of icmpInMsgs with time
icmpInMsgs(/5s)
30 k
20 k
10 k
0
11:50
11:55
12:00
12:05
🟥 Average change in icmpInMsgs per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_Ping_Flood - Rate of change of icmpOutEchoReps with time
icmpOutEchoReps(/5s)
20 k
10 k
0
11:50
11:55
12:00
12:05
🟥 Average change in icmpOutEchoReps per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_Ping_Flood - Rate of change of icmpOutMsgs with time
icmpOutMsgs(/5s)
20 k
10 k
0
11:50
11:55
12:00
12:05
🟥 Average change in icmpOutMsgs per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_Ping_Flood - Rate of change of ipInDelivers with time
ipInDelivers(/5s)
30 k
20 k
10 k
0
11:50
11:55
12:00
12:05
🟥 Average change in ipInDelivers per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_Ping_Flood - Rate of change of ipInReceives with time
ipInReceives(/5s)
30 k
20 k
10 k
0
11:50
11:55
12:00
12:05
🟥 Average change in ipInReceives per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_Ping_Flood - Rate of change of ipOutRequests with time
ipOutRequests(/5s)
20 k
10 k
0
11:50
11:55
12:00
12:05
🟥 Average change in ipOutRequests per five seconds
RRDTOOL / TOBI OETIKER

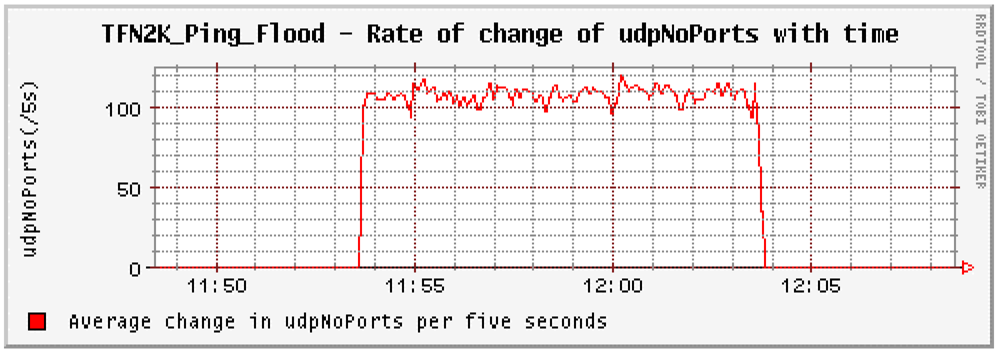
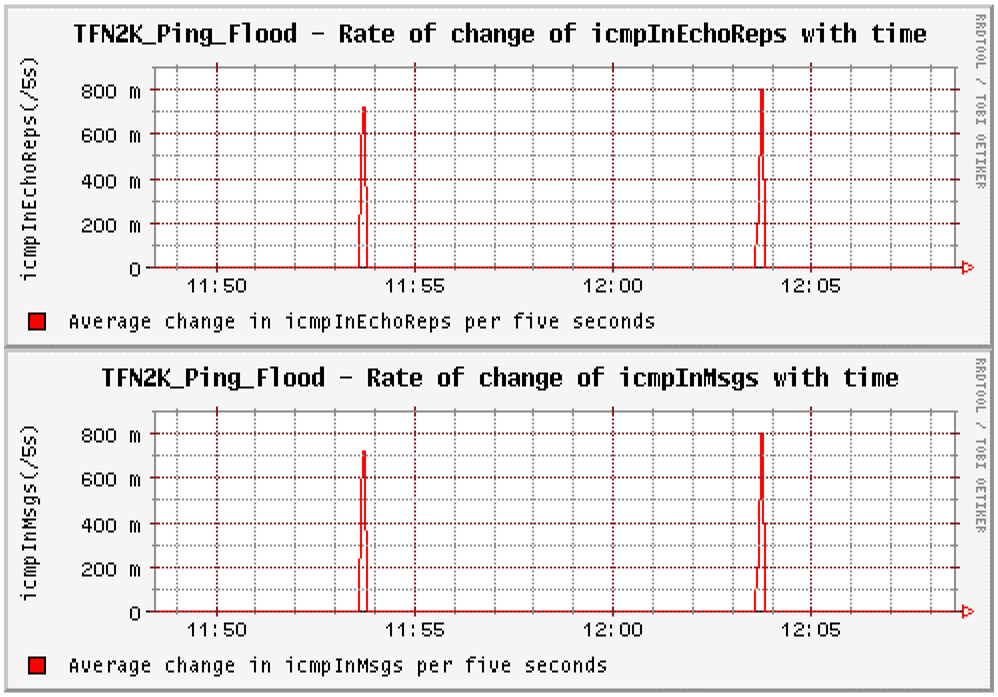
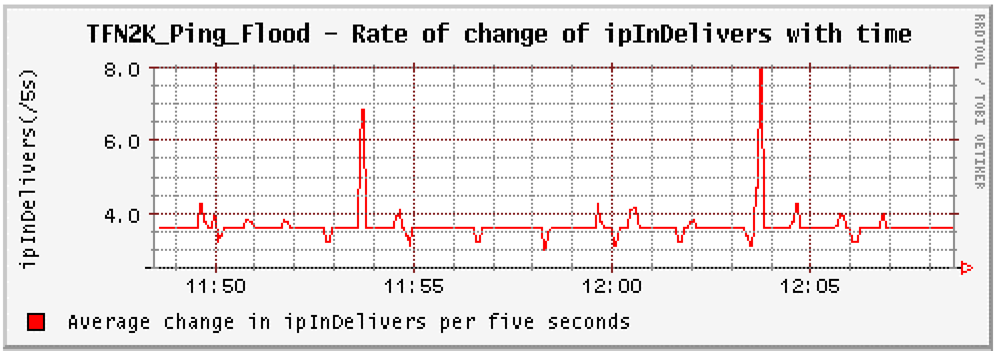


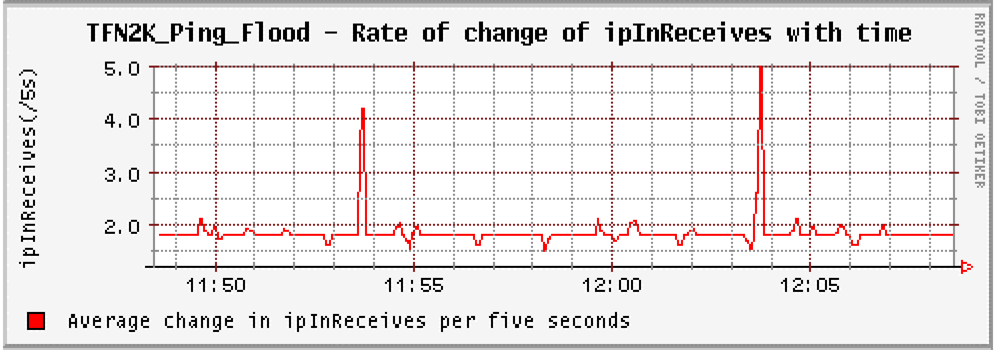
Figure 18. TFN2K Ping Flood Attack - Changes In Target MIB Variables.

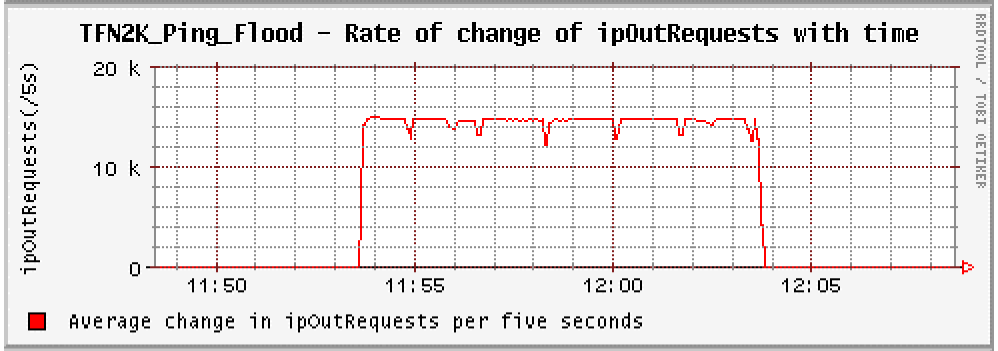
b. Changes in the Attacker-1 MIB Variables

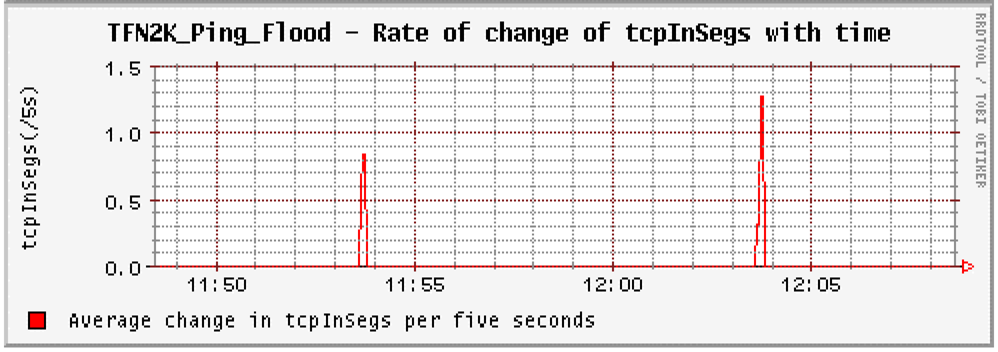
Figure 19 shows the average changes in the attacker1 MIB variables per five-second interval when under a TFN2K ping-flood attack.




TFN2K_Ping_Flood - Rate of change of ipInDelivers with time
ipInDelivers(/5s)
8.0
6.0
4.0
11:50
11:55
12:00
12:05
🟥 Average change in ipInDelivers per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_Ping_Flood - Rate of change of ipInReceives with time
ipInReceives(/5s)
5.0
4.0
3.0
2.0
11:50
11:55
12:00
12:05
🟥 Average change in ipInReceives per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_Ping_Flood - Rate of change of ipOutRequests with time
ipOutRequests(/5s)
20 k
10 k
0
11:50
11:55
12:00
12:05
🟥 Average change in ipOutRequests per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_Ping_Flood - Rate of change of tcpInSegs with time
tcpInSegs(/5s)
1.5
1.0
0.5
0.0
11:50
11:55
12:00
12:05
🟥 Average change in tcpInSegs per five seconds
RRDTOOL / TOBI OETIKER

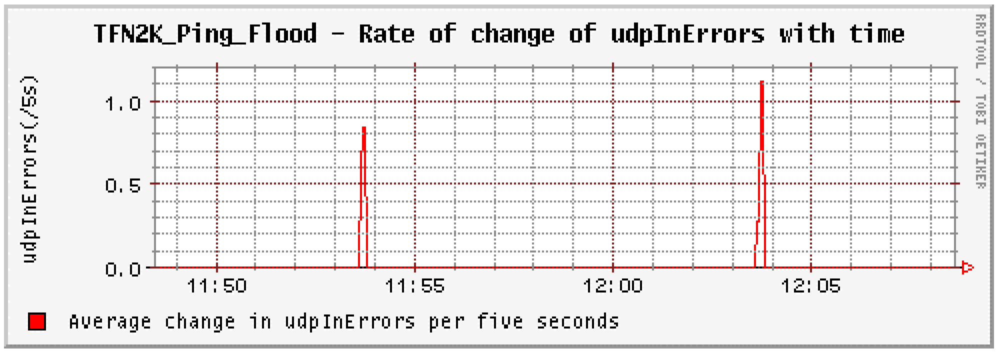
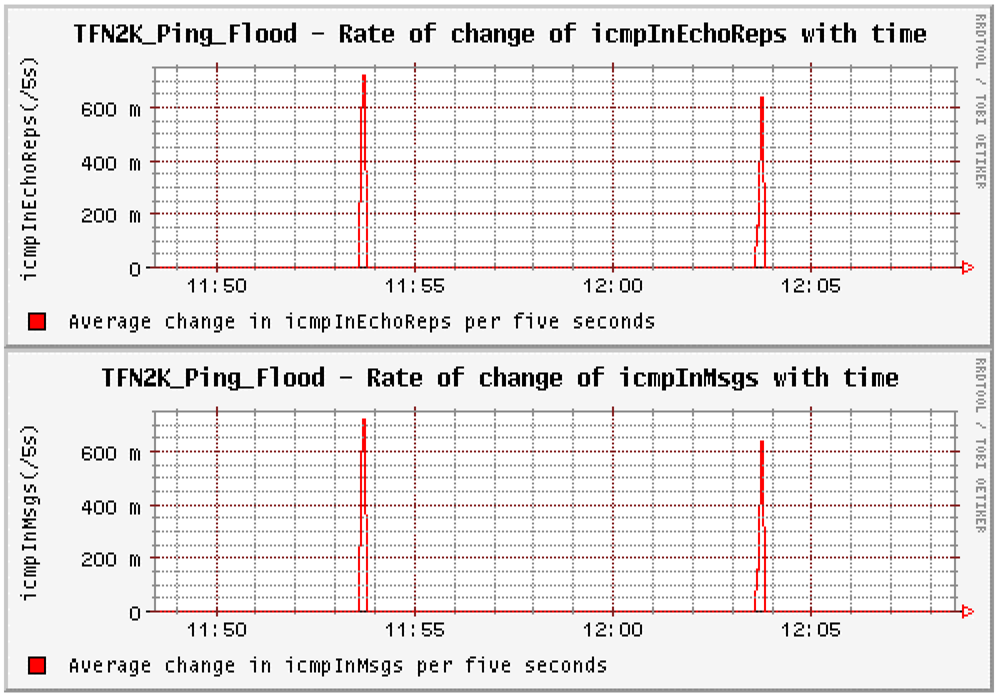
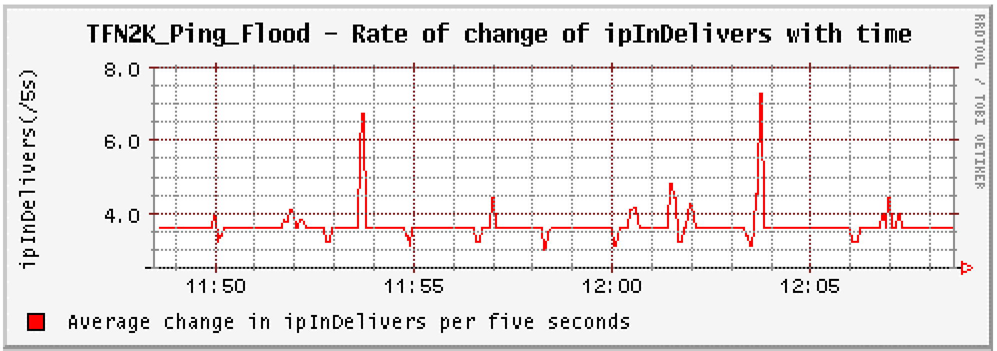


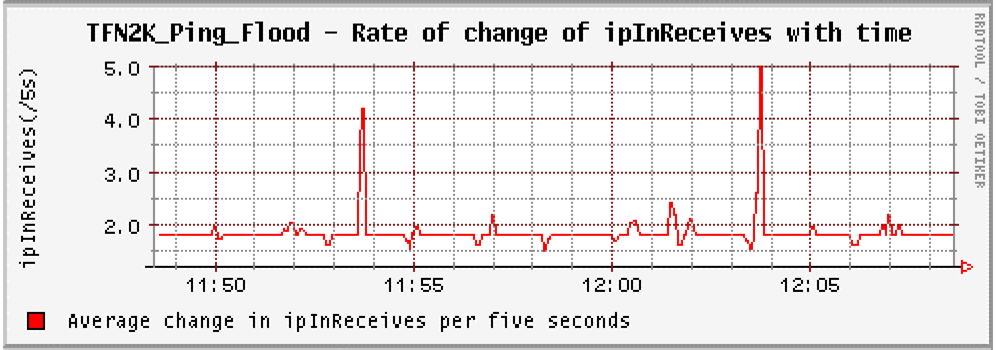
Figure 19. TFN2K Ping Flood Attack - Changes In Attacker1 MIB Variables.

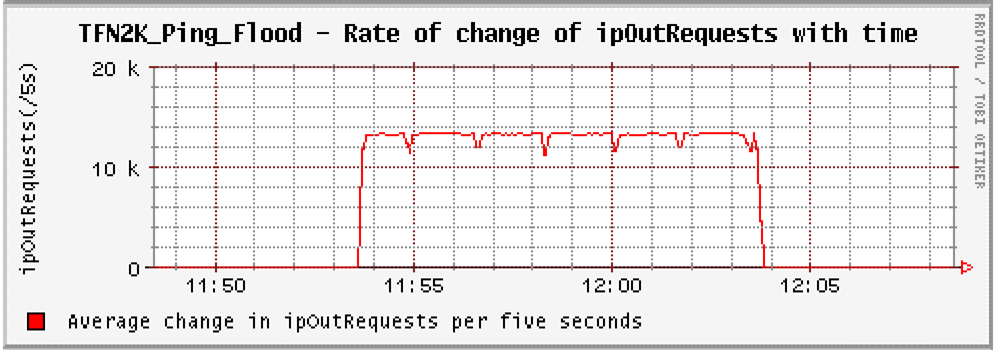
c. Changes in the Attacker-2 MIB Variables

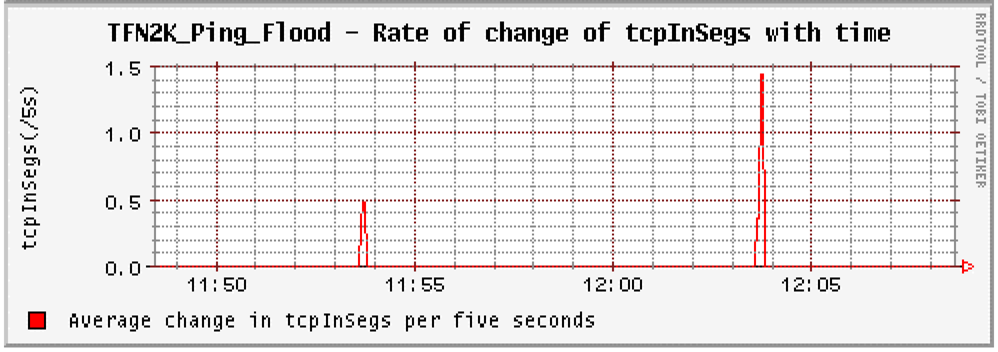
Figure 20 shows the average changes in the attacker2 MIB variables per five-second interval when under a TFN2K ping-flood attack.




TFN2K_Ping_Flood - Rate of change of ipInDelivers with time
ipInDelivers(/5s)
8.0
6.0
4.0
11:50
11:55
12:00
12:05
🟥 Average change in ipInDelivers per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_Ping_Flood - Rate of change of ipInReceives with time
ipInReceives(/5s)
5.0
4.0
3.0
2.0
11:50
11:55
12:00
12:05
🟥 Average change in ipInReceives per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_Ping_Flood - Rate of change of ipOutRequests with time
ipOutRequests(/5s)
20 k
10 k
0
11:50
11:55
12:00
12:05
🟥 Average change in ipOutRequests per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_Ping_Flood - Rate of change of tcpInSegs with time
tcpInSegs(/5s)
1.5
1.0
0.5
0.0
11:50
11:55
12:00
12:05
🟥 Average change in tcpInSegs per five seconds
RRDTOOL / TOBI OETIKER

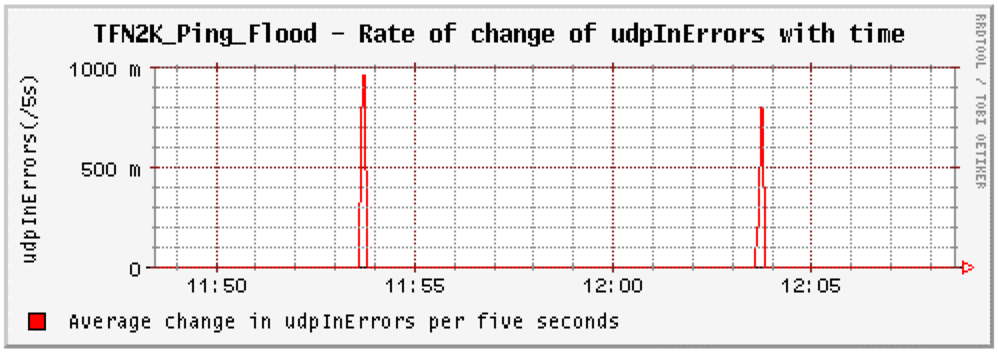


Figure 20. TFN2K Ping Flood Attack - Changes In Attacker2 MIB Variables.

2. SYN Flood

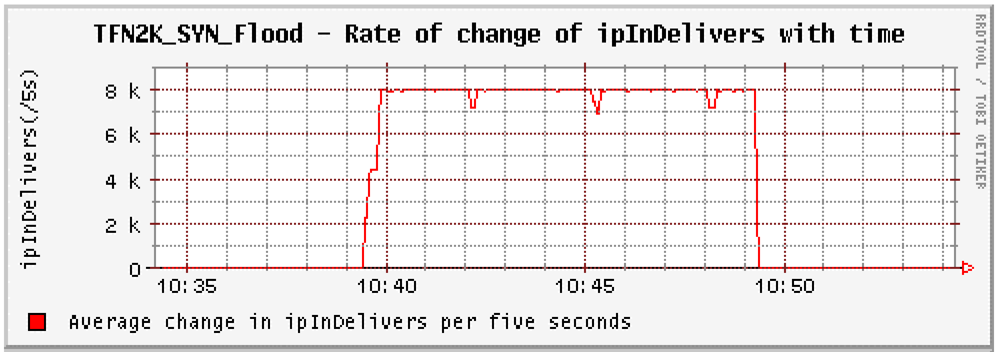
The affected variables at the target and at the attacker in a TFN2K syn-flood attack are shown in table 6.

Sl. No.	Target	Attacker
1.	**ip.ipInDelivers.0**	icmp.icmpInEchoReps.0
2.	**ip.ipInReceives.0**	icmp.icmpInMsgs.0
3.	ip.ipOutRequests.0	ip.ipInReceives.0
4.	**tcp.tcpInSegs.0**	ip.ipInDelivers.0
5.	**tcp.tcpInErrs.0**	**ip.ipOutRequests.0**
6.	**tcp.tcpOutRsts.0**	**tcp.tcpInSegs.0**
7.	**tcp.tcpOutSegs.0**	udp.udpInErrors.0

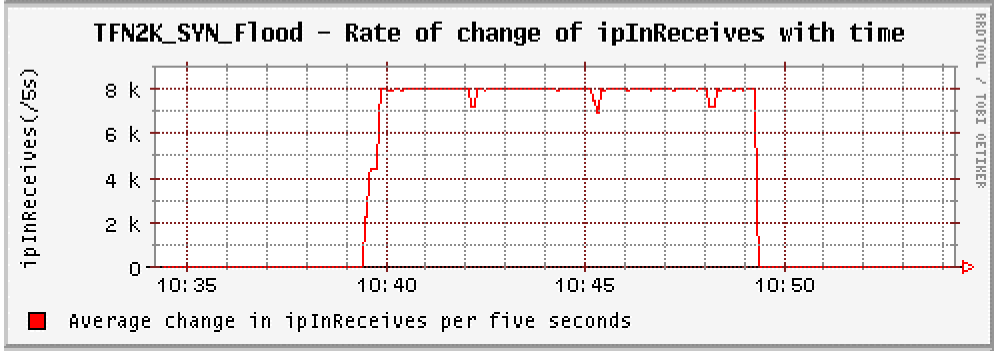
Table 6. Affected MIB Variables In TFN2K Syn Flood Attack.

a. Changes in the Target MIB Variables

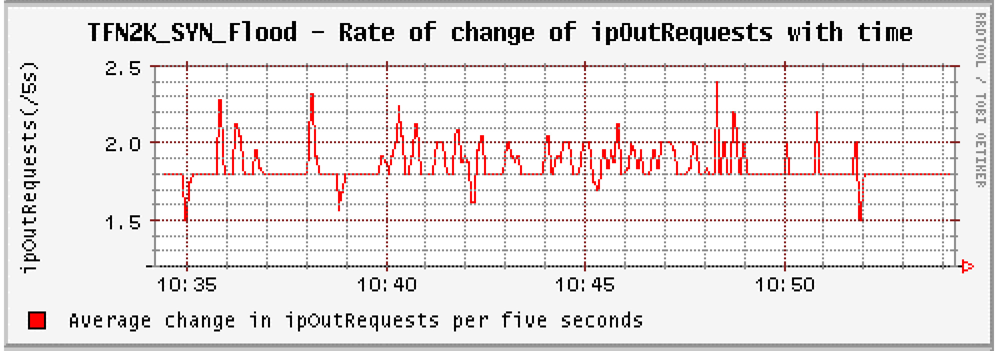
Figure 21 below shows the average change in the target MIB variables per five-second interval for a TFN2K syn-flood attack.



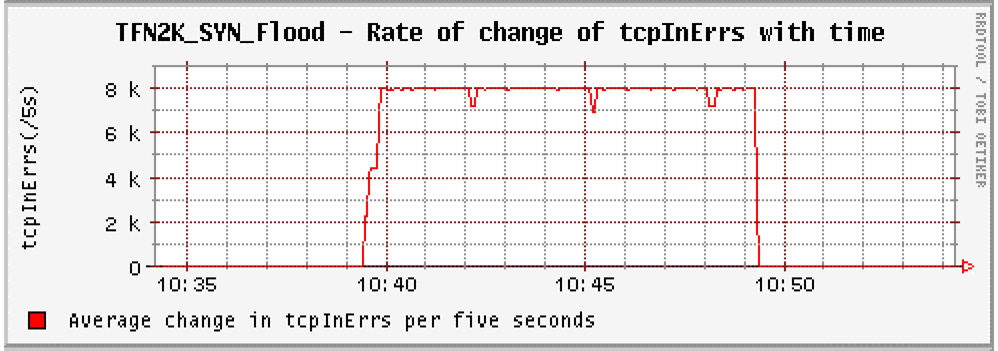
TFN2K_SYN_Flood - Rate of change of ipInDelivers with time
ipInDelivers(/5s)
8 k
6 k
4 k
2 k
0
10:35
10:40
10:45
10:50
🟥 Average change in ipInDelivers per five seconds
RRDTOOL / TOBI OETIKER



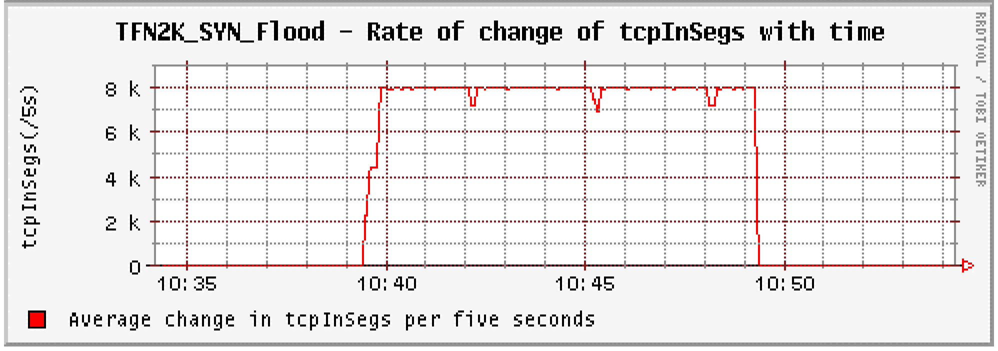
TFN2K_SYN_Flood - Rate of change of ipInReceives with time
ipInReceives(/5s)
8 k
6 k
4 k
2 k
0
10:35
10:40
10:45
10:50
🟥 Average change in ipInReceives per five seconds
RRDTOOL / TOBI OETIKER



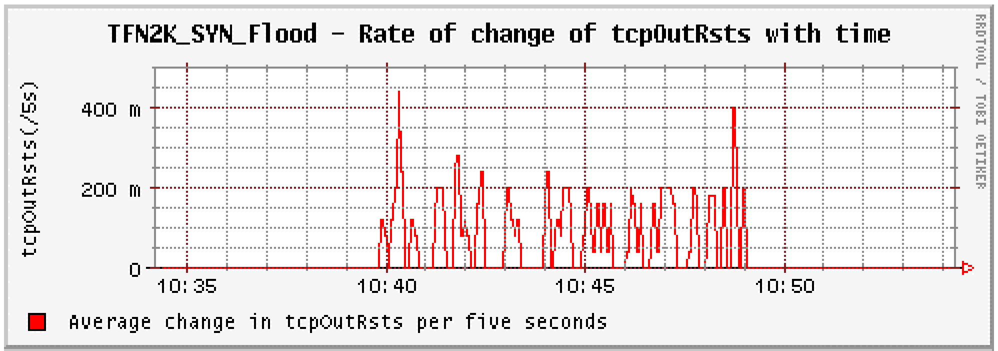
TFN2K_SYN_Flood - Rate of change of ipOutRequests with time
ipOutRequests(/5s)
2.5
2.0
1.5
10:35
10:40
10:45
10:50
🟥 Average change in ipOutRequests per five seconds
RRDTOOL / TOBI OETIKER



TFN2K_SYN_Flood - Rate of change of tcpInErrs with time
tcpInErrs(/5s)
8 k
6 k
4 k
2 k
0
10:35
10:40
10:45
10:50
🟥 Average change in tcpInErrs per five seconds
RRDTOOL / TOBI OETIKER



TFN2K_SYN_Flood - Rate of change of tcpInSegs with time
tcpInSegs(/5s)
8 k
6 k
4 k
2 k
0
10:35
10:40
10:45
10:50
🟥 Average change in tcpInSegs per five seconds
RRDTOOL / TOBI OETIKER



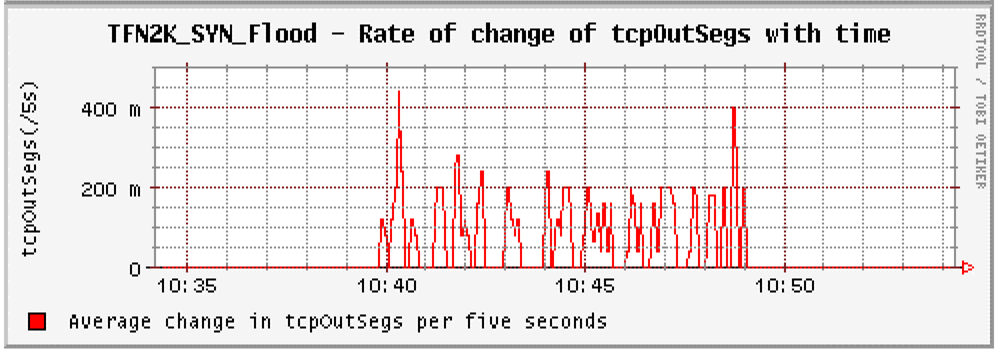
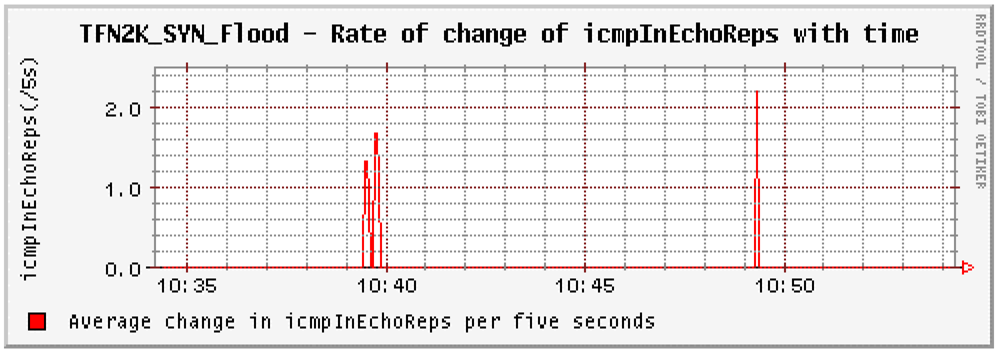
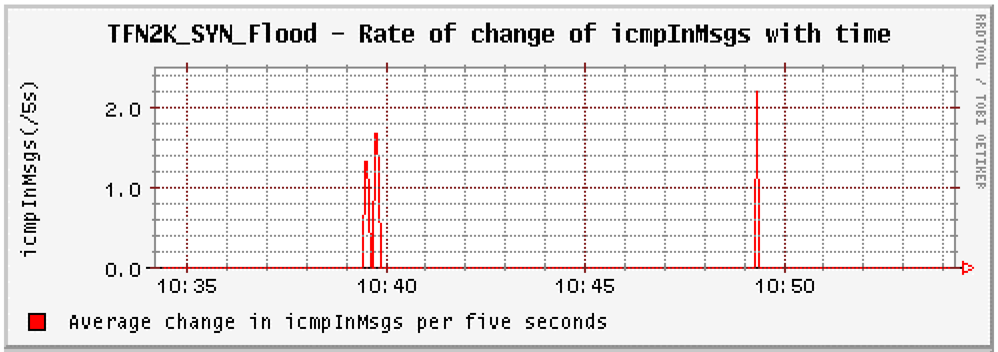


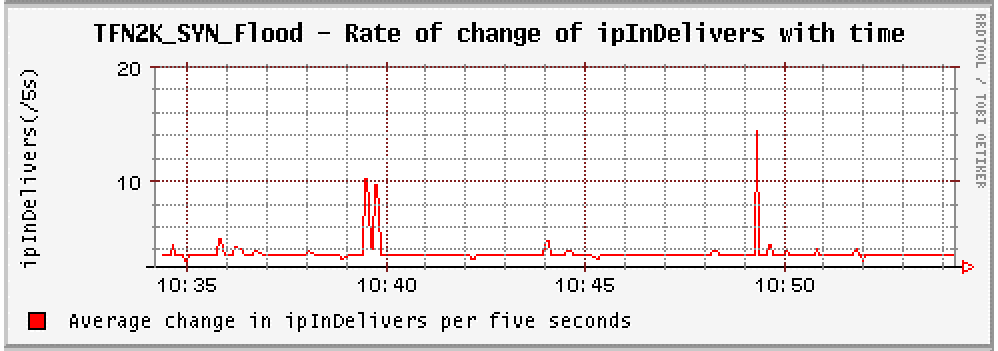
Figure 21. TFN2K Syn Flood Attack - Changes In Target MIB Variables.

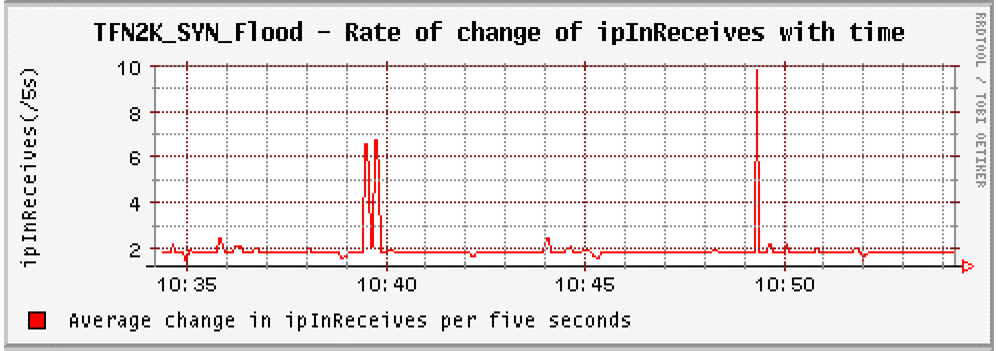
b. Changes in the Attacker-1 MIB Variables

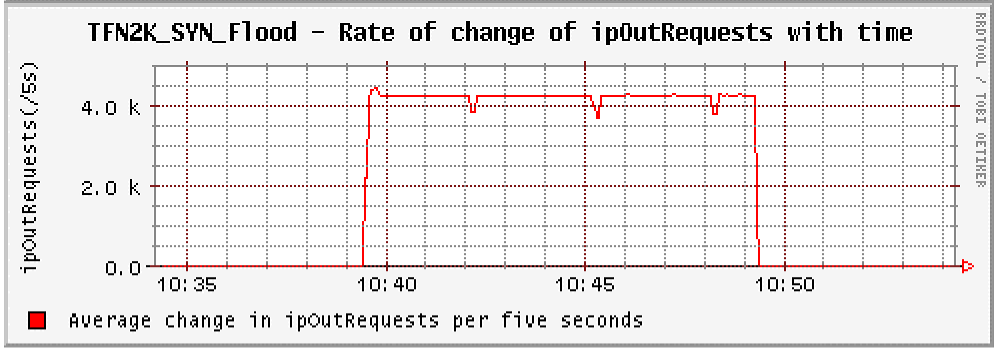
Figure 22 shows the average changes in the attacker1 MIB variables per five-second interval when under a TFN2K syn-flood attack.




TFN2K_SYN_Flood - Rate of change of icmpInMsgs with time
icmpInMsgs(/5s)
2.0
1.0
0.0
10:35
10:40
10:45
10:50
Average change in icmpInMsgs per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_SYN_Flood - Rate of change of ipInDelivers with time
ipInDelivers(/5s)
20
10
10:35
10:40
10:45
10:50
Average change in ipInDelivers per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_SYN_Flood - Rate of change of ipInReceives with time
ipInReceives(/5s)
10
8
6
4
2
10:35
10:40
10:45
10:50
Average change in ipInReceives per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_SYN_Flood - Rate of change of ipOutRequests with time
ipOutRequests(/5s)
4.0 k
2.0 k
0.0
10:35
10:40
10:45
10:50
Average change in ipOutRequests per five seconds
RRDTOOL / TOBI OETIKER

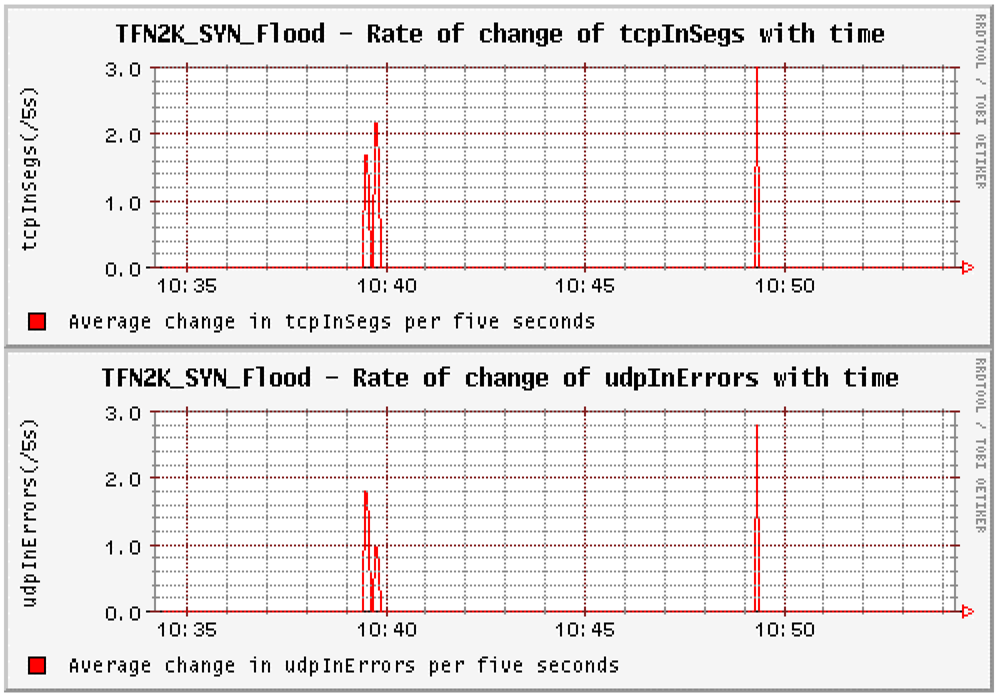
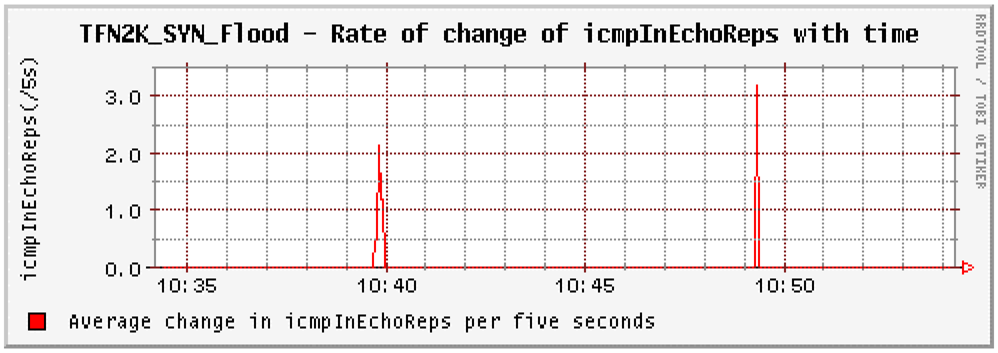
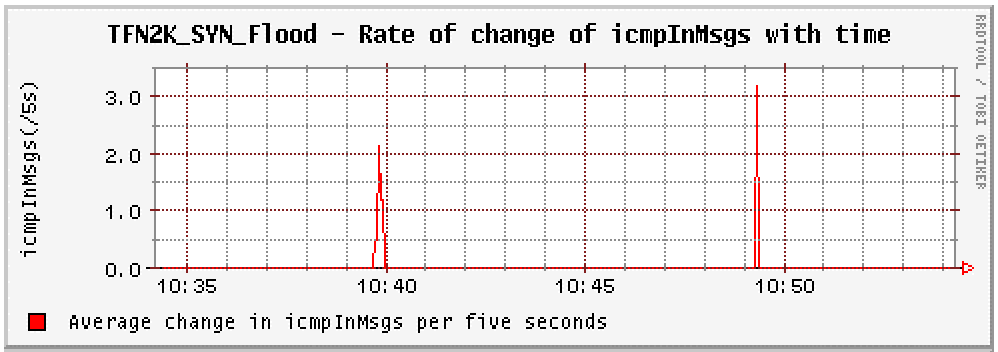


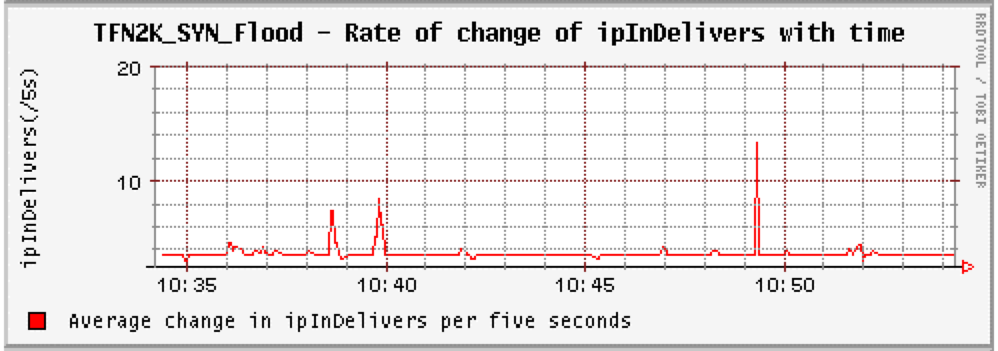
Figure 22. TFN2K Syn Flood Attack - Changes In Attacker1 MIB Variables.

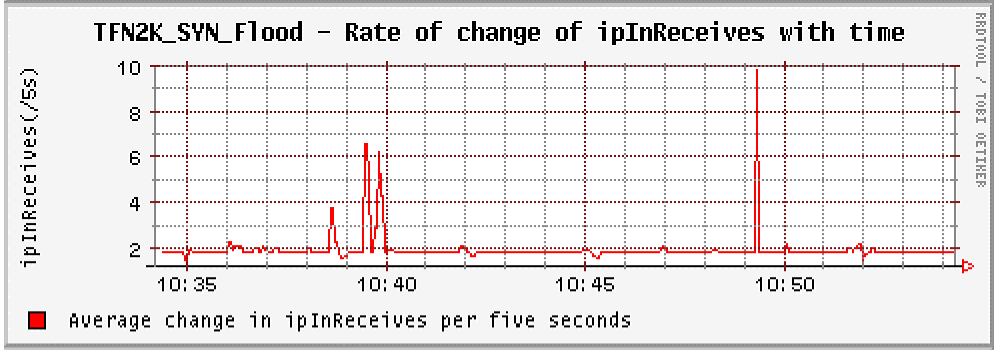
c. Changes in the Attacker-2 MIB Variables

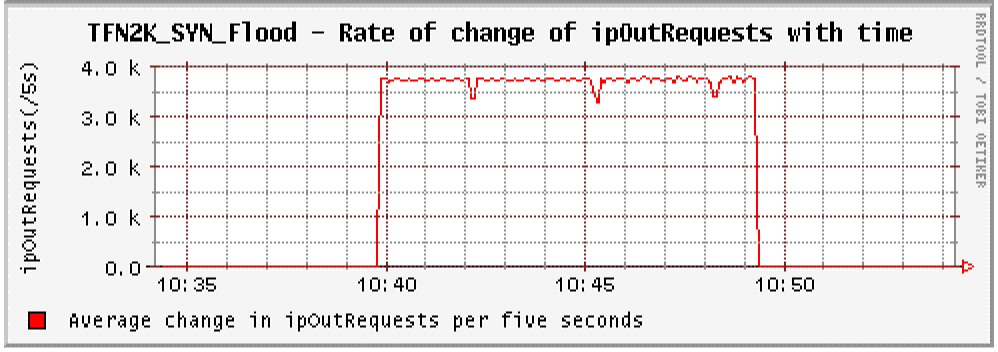
Figure 13 shows the average changes in the attacker2 MIB variables per five-second interval when under a TFN2K syn-flood attack.

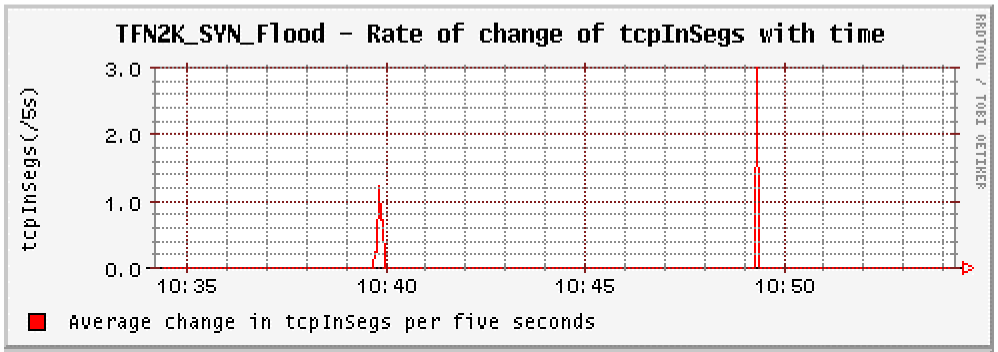



TFN2K_SYN_Flood - Rate of change of icmpInMsgs with time
icmpInMsgs(/5s)
3.0
2.0
1.0
0.0
10:35
10:40
10:45
10:50
🟥 Average change in icmpInMsgs per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_SYN_Flood - Rate of change of ipInDelivers with time
ipInDelivers(/5s)
20
10
10:35
10:40
10:45
10:50
🟥 Average change in ipInDelivers per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_SYN_Flood - Rate of change of ipInReceives with time
ipInReceives(/5s)
10
8
6
4
2
10:35
10:40
10:45
10:50
🟥 Average change in ipInReceives per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_SYN_Flood - Rate of change of ipOutRequests with time
ipOutRequests(/5s)
4.0 k
3.0 k
2.0 k
1.0 k
0.0
10:35
10:40
10:45
10:50
🟥 Average change in ipOutRequests per five seconds
RRDTOOL / TOBI OETIKER



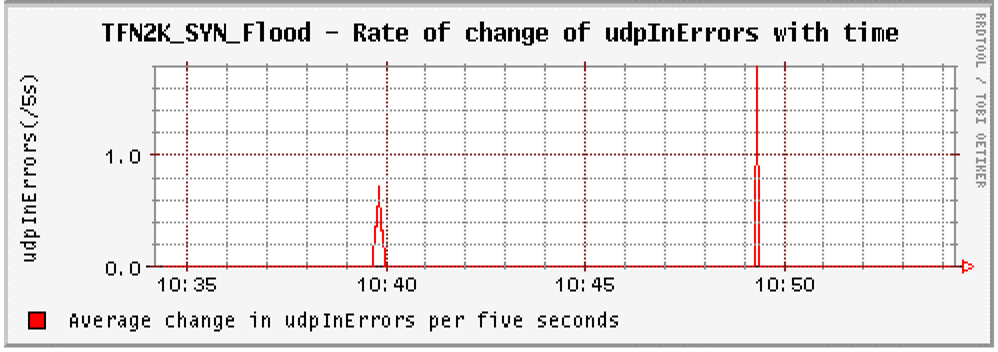


Figure 23. TFN2K Syn Flood Attack - Changes In Attacker2 MIB Variables.

3. UDP Flood

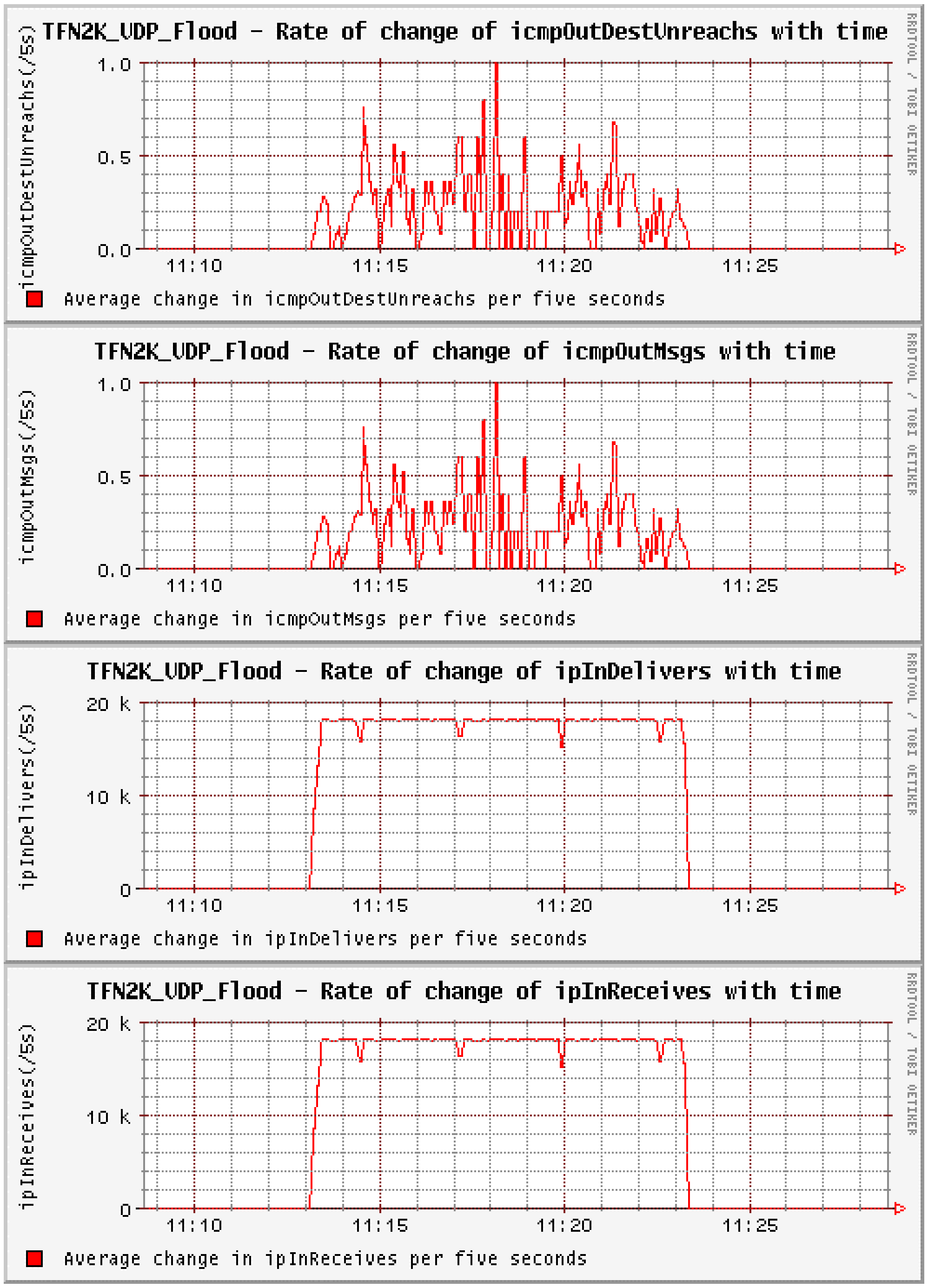
The affected variables at the target and at the attacker in a TFN2K UDP-Flood attack are shown in table 7.

Sl. No.	Target	Attacker
1.	icmp.icmpOutDestUnreachs.0	icmp.icmpInEchoReps.0
2.	icmp.icmpOutMsgs.0	icmp.icmpInMsgs.0
3.	**ip.ipInDelivers.0**	ip.ipInReceives.0
4.	**ip.ipInReceives.0**	ip.ipInDelivers.0
5.	ip.ipOutRequests.0	**ip.ipOutRequests.0**
6.	**udp.udpInErrors.0**	tcp.tcpInSegs.0
7.	udp.udpNoPorts.0	udp.udpInErrors.0

Table 7. Affected MIB Variables In TFN2K UDP Flood Attack.

a. *Changes in the Target MIB Variables*

Figure 24 below shows the average change in the target MIB variables per five-second interval for a TFN2K UDP-flood attack.



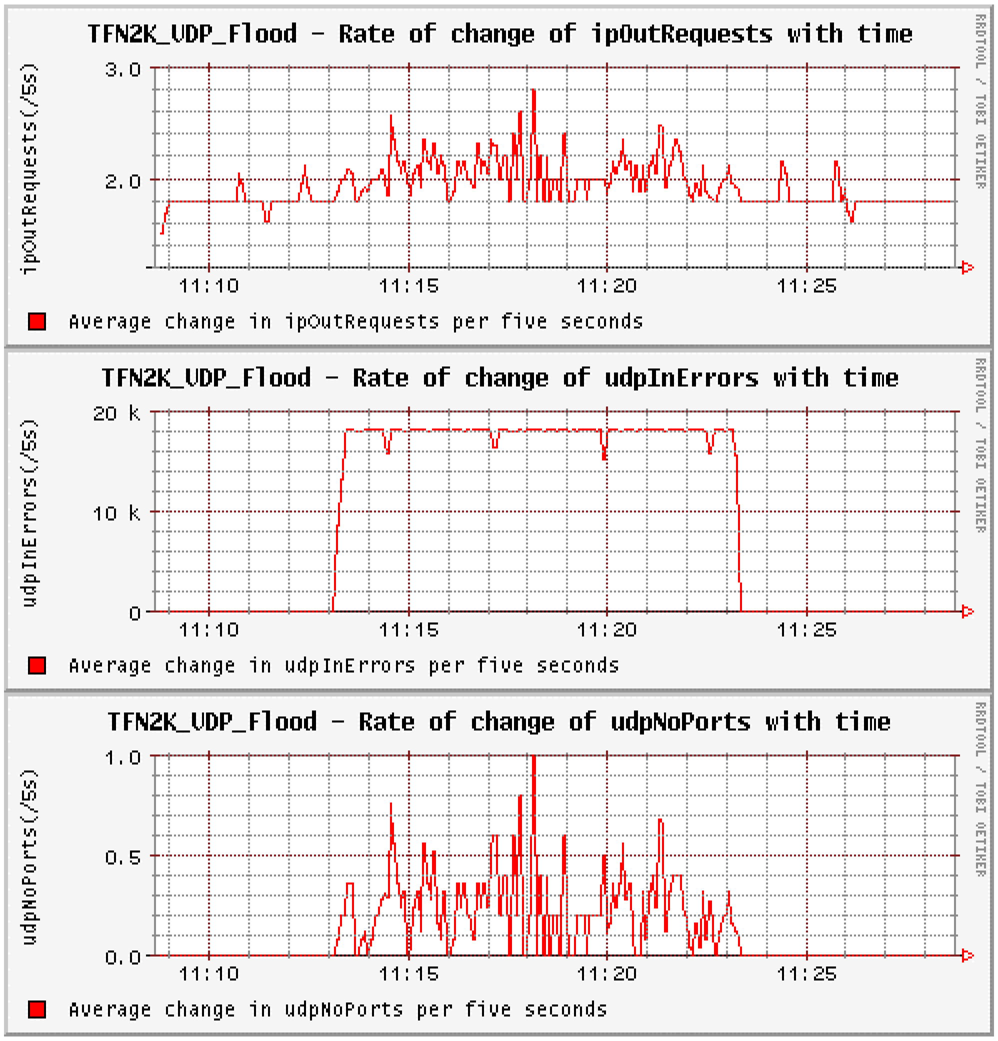
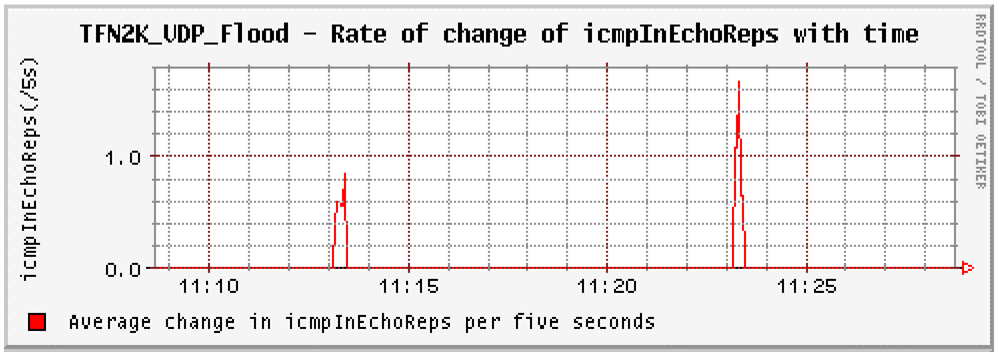
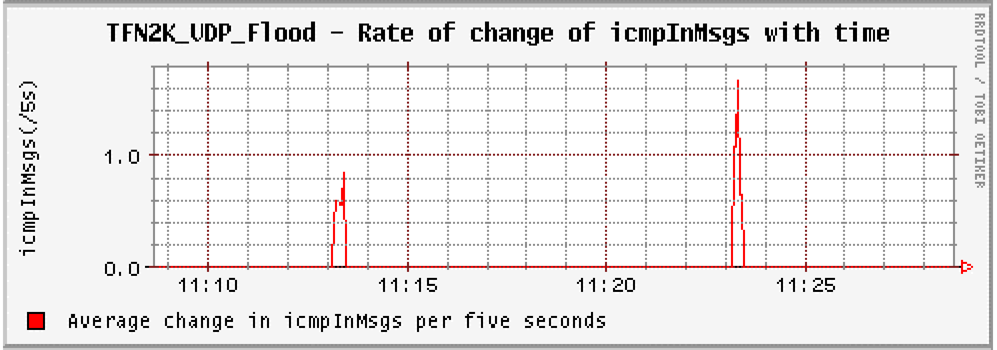


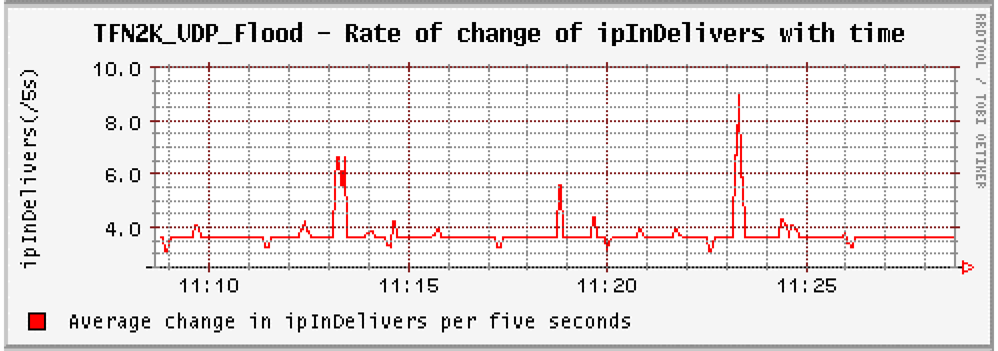
Figure 24. TFN2K UDP Flood Attack - Changes In Target MIB Variables.

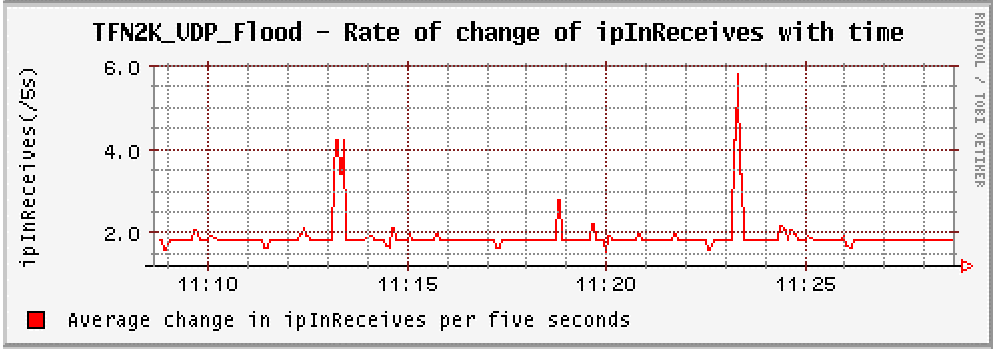
b. Changes in the Attacker-1 MIB Variables

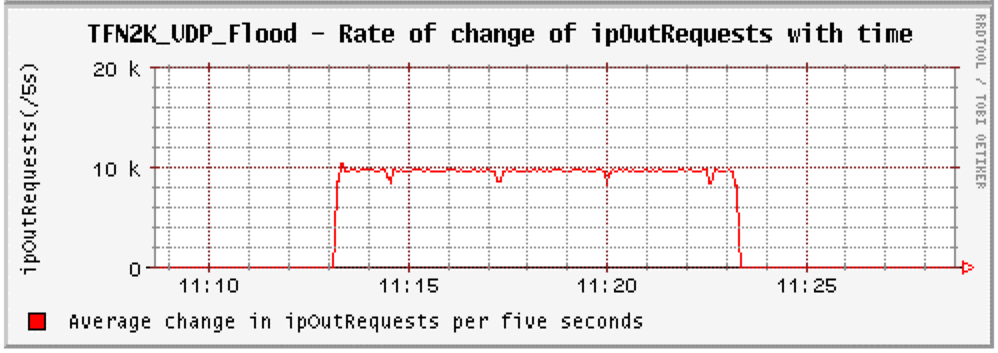
Figure 25 shows the average changes in the attacker1 MIB variables per five-second interval when under a TFN2K UDP-flood attack.


TFN2K_UDP_Flood - Rate of change of icmpInEchoReps with time
icmpInEchoReps(/5s)
1.0
0.0
11:10
11:15
11:20
11:25
Average change in icmpInEchoReps per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_UDP_Flood - Rate of change of icmpInMsgs with time
icmpInMsgs(/5s)
1.0
0.0
11:10
11:15
11:20
11:25
Average change in icmpInMsgs per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_UDP_Flood - Rate of change of ipInDelivers with time
ipInDelivers(/5s)
10.0
8.0
6.0
4.0
11:10
11:15
11:20
11:25
Average change in ipInDelivers per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_UDP_Flood - Rate of change of ipInReceives with time
ipInReceives(/5s)
6.0
4.0
2.0
11:10
11:15
11:20
11:25
Average change in ipInReceives per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_UDP_Flood - Rate of change of ipOutRequests with time
ipOutRequests(/5s)
20 k
10 k
0
11:10
11:15
11:20
11:25
Average change in ipOutRequests per five seconds
RRDTOOL / TOBI OETIKER

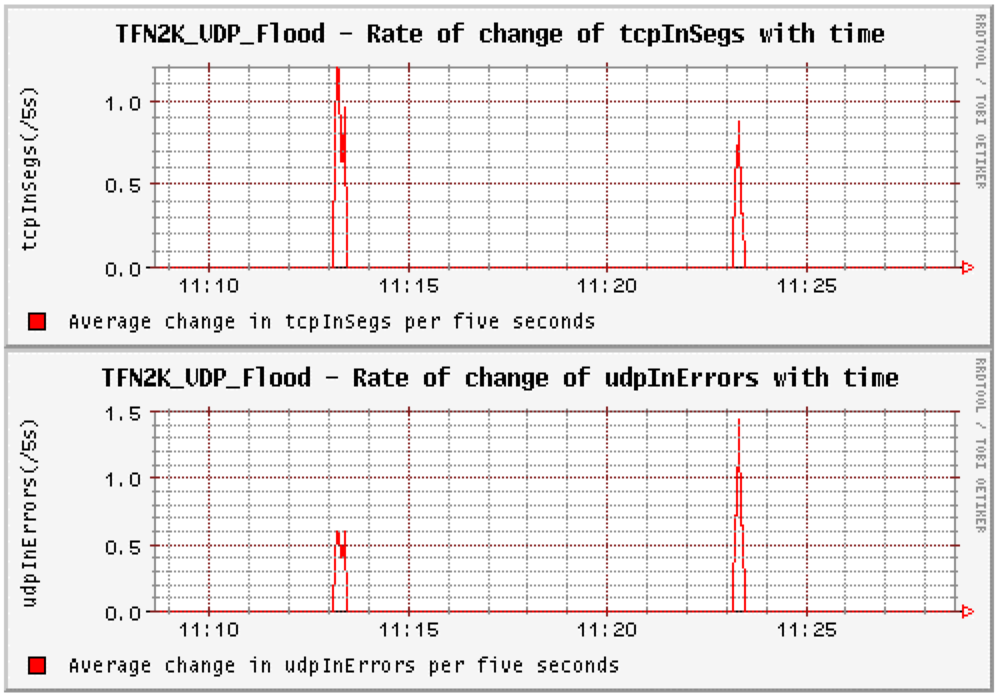
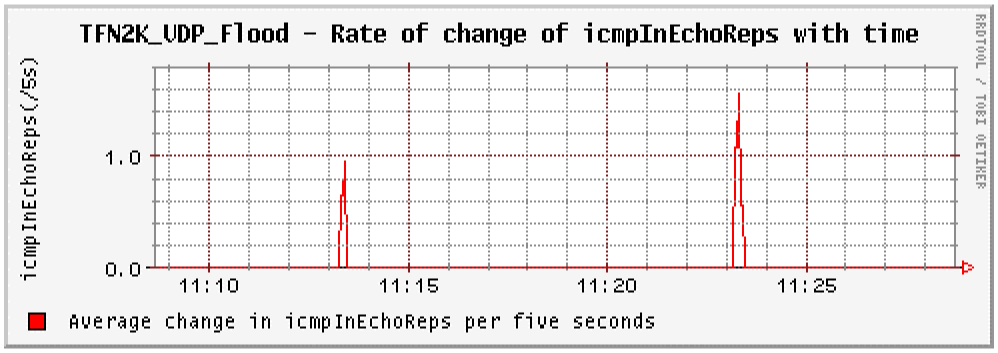
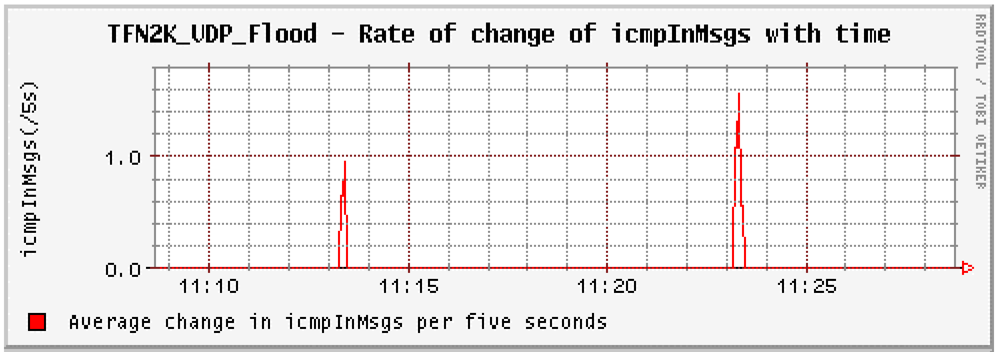


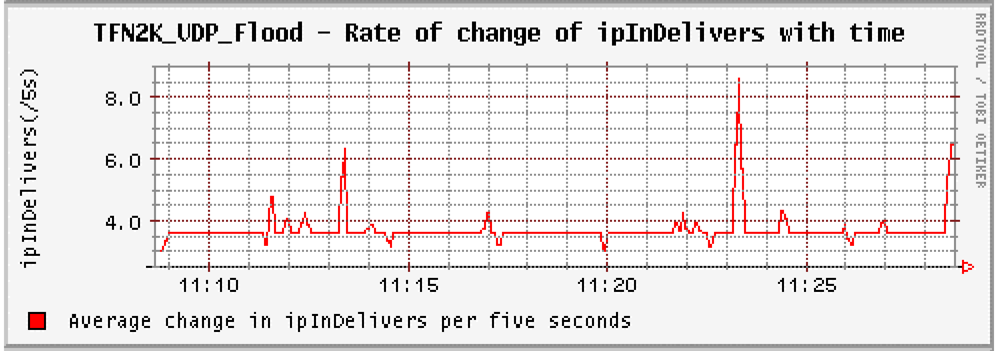
Figure 25. TFN2K UDP Flood Attack - Changes In Attacker1 MIB Variables.

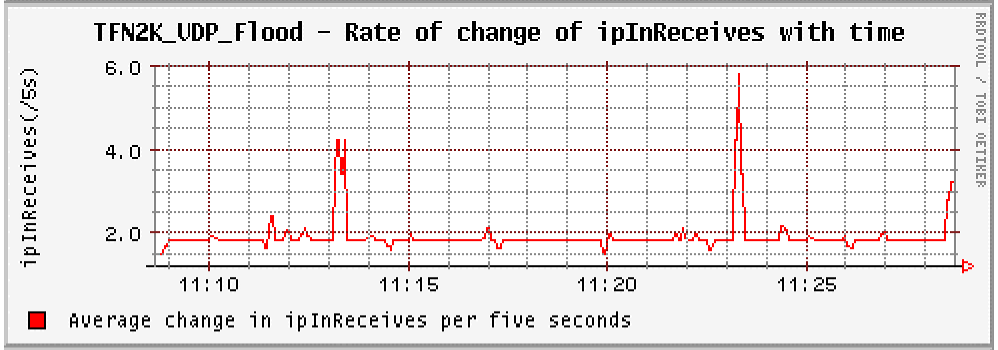
c. Changes in the Attacker-2 MIB Variables

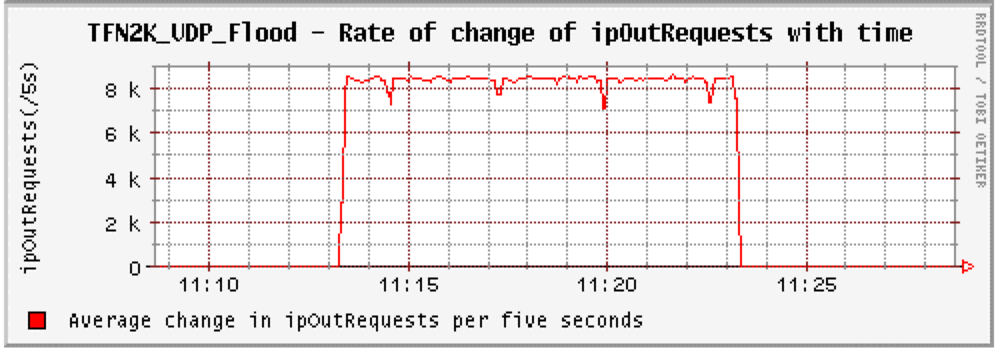
Figure 26 shows the average changes in the attacker2 MIB variables per five-second interval when under a TFN2K UDP-flood attack.




TFN2K_UDP_Flood - Rate of change of icmpInMsgs with time
icmpInMsgs(/5s)
1.0
0.0
11:10
11:15
11:20
11:25
Average change in icmpInMsgs per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_UDP_Flood - Rate of change of ipInDelivers with time
ipInDelivers(/5s)
8.0
6.0
4.0
11:10
11:15
11:20
11:25
Average change in ipInDelivers per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_UDP_Flood - Rate of change of ipInReceives with time
ipInReceives(/5s)
6.0
4.0
2.0
11:10
11:15
11:20
11:25
Average change in ipInReceives per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_UDP_Flood - Rate of change of ipOutRequests with time
ipOutRequests(/5s)
8 k
6 k
4 k
2 k
0
11:10
11:15
11:20
11:25
Average change in ipOutRequests per five seconds
RRDTOOL / TOBI OETIKER

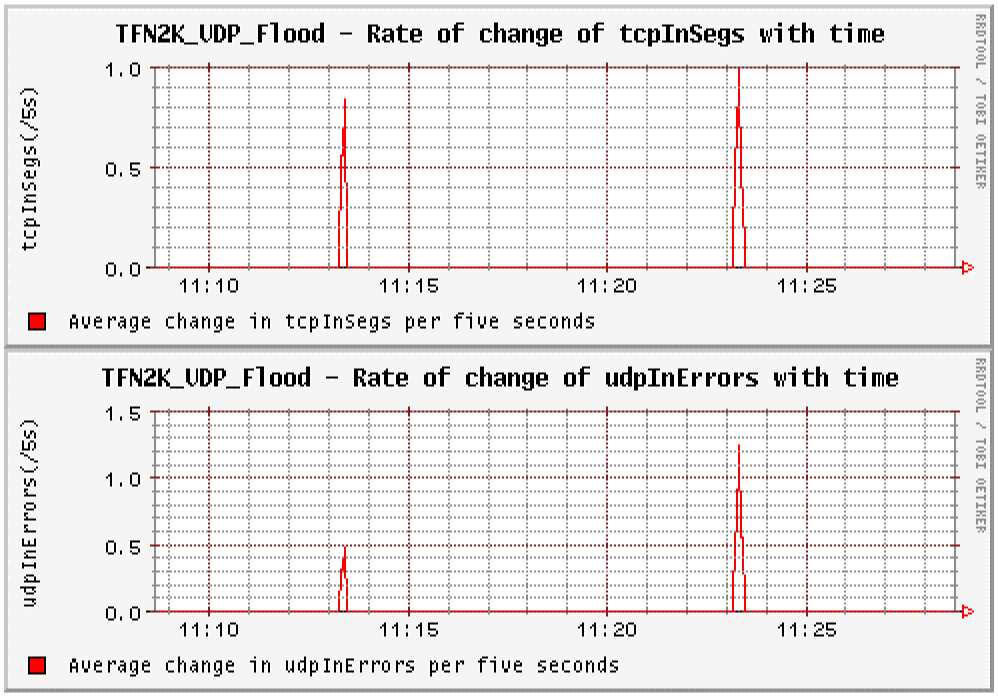


Figure 26. TFN2K UDP Flood Attack - Changes In Attacker2 MIB Variables.

4. Targa Flood

The affected variables at the target and at the attacker in a TFN2K Targa-Flood attack are shown in table 8.

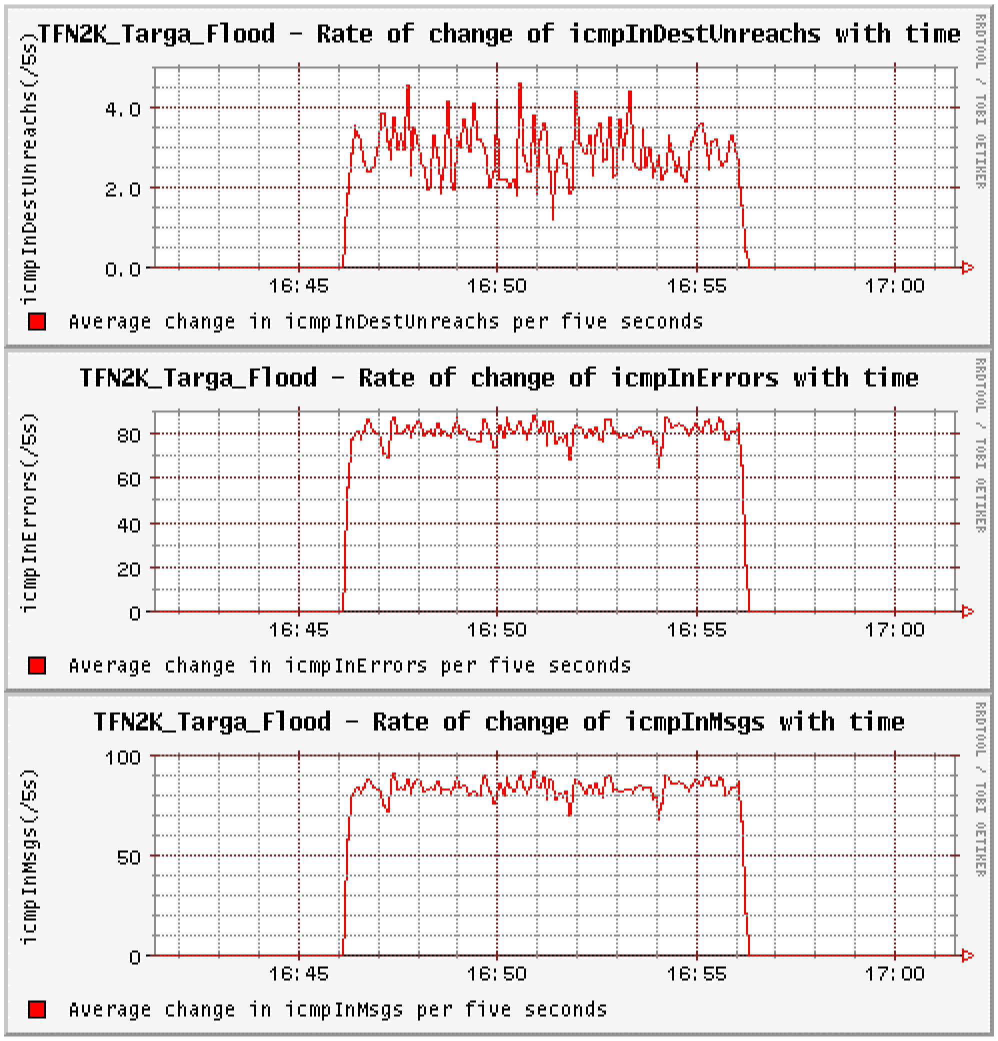
Sl. No.	Target	Attacker
1.	**icmp.icmpInDestUnreachs.0**	icmp.icmpInEchoReps.0
2.	**icmp.icmpInErrors.0**	icmp.icmpInMsgs.0
3.	**icmp.icmpInMsgs.0**	ip.ipInDelivers.0
4.	**icmp.icmpOutDestUnreachs.0**	ip.ipInReceives.0
5.	**icmp.icmpOutMsgs.0**	**ip.ipOutRequests.0**
6.	**ip.ipInDelivers.0**	tcp.tcpInSegs.0
7.	**ip.ipInReceives.0**	udp.udpInErrors.0
8.	**ip.ipInUnknownProtos.0**	udp.udpOutDatagrams.0
9.	**ip.ipOutRequests.0**	
10.	**ip.ipReasmFails.0**	
11.	**ip.ipReasmReqds.0**	
12.	tcp.tcpInErrs.0	

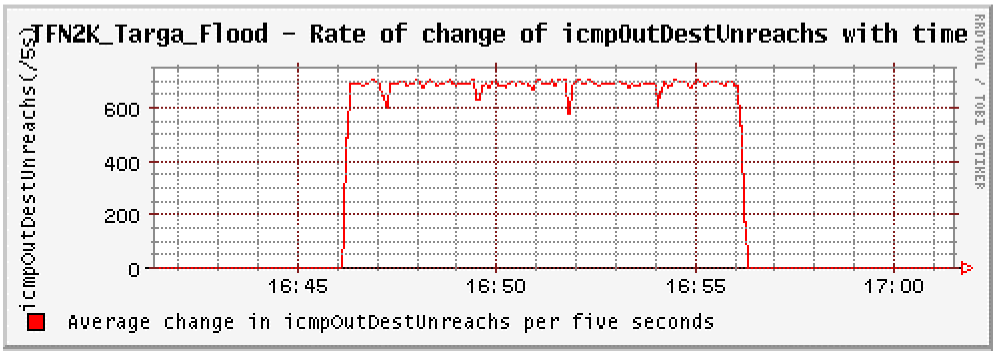
13.	tcp.tcpInSegs.0	
14.	udp.udpInErrors.0	
15.	**udp.udpNoPorts.0**	
16.	udp.udpOutDatagrams.0	

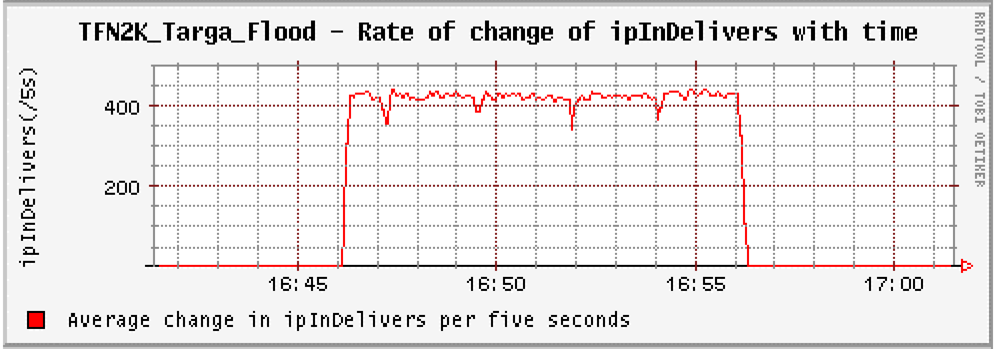
Table 8. Affected MIB Variables In TFN2K Targa Flood Attack.

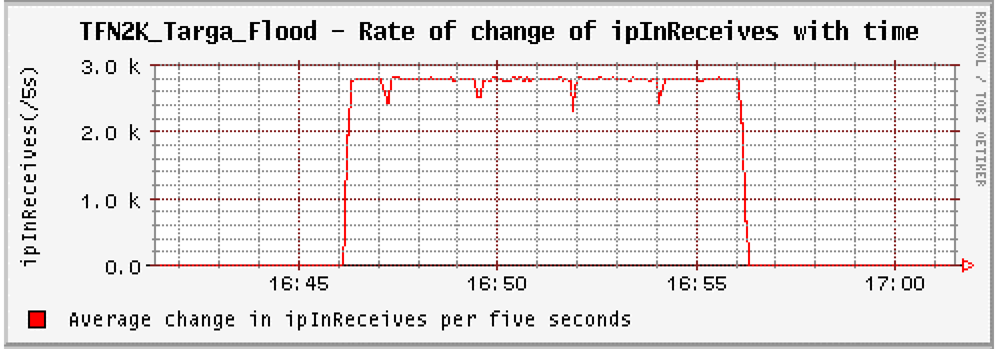
a. Changes in the Target MIB Variables

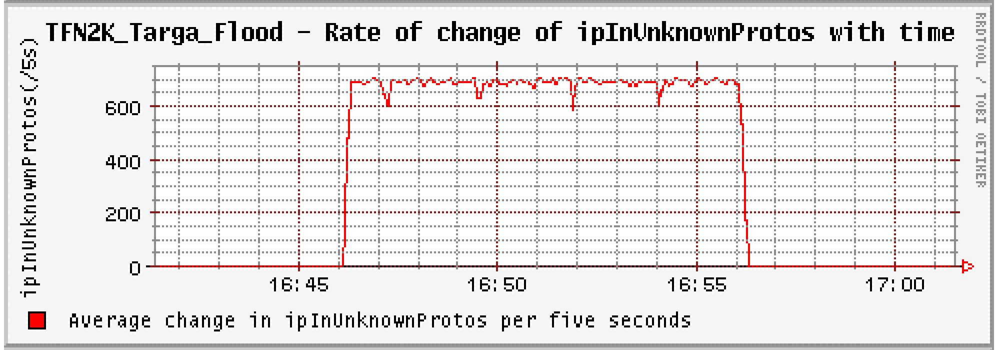
Figure 27 below shows the average change in the target MIB variables per five-second interval for a TFN2K targa-flood attack.

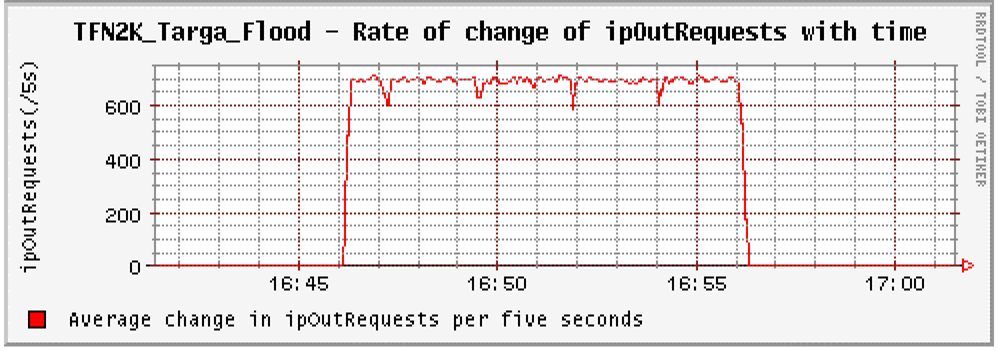


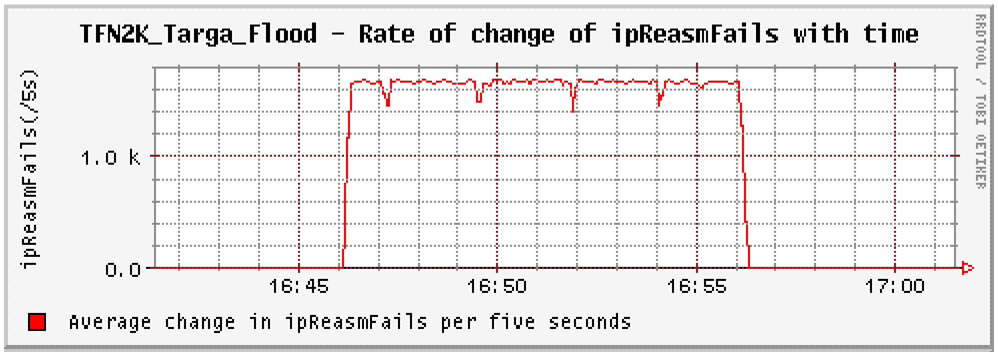

TFN2K_Targa_Flood - Rate of change of icmpOutDestUnreachs with time
icmpOutDestUnreachs(/5s)
600
400
200
0
16:45
16:50
16:55
17:00
🟥 Average change in icmpOutDestUnreachs per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_Targa_Flood - Rate of change of ipInDelivers with time
ipInDelivers(/5s)
400
200
16:45
16:50
16:55
17:00
🟥 Average change in ipInDelivers per five seconds
RRDTOOL / TOBI OETIKER

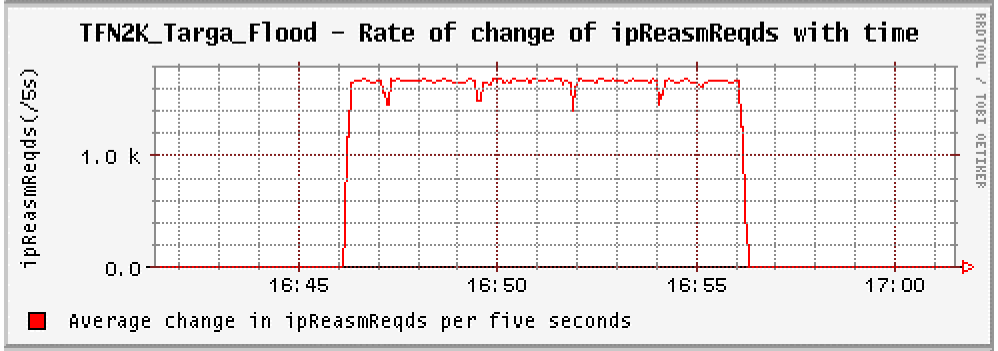

TFN2K_Targa_Flood - Rate of change of ipInReceives with time
ipInReceives(/5s)
3.0 k
2.0 k
1.0 k
0.0
16:45
16:50
16:55
17:00
🟥 Average change in ipInReceives per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_Targa_Flood - Rate of change of ipInUnknownProtos with time
ipInUnknownProtos(/5s)
600
400
200
0
16:45
16:50
16:55
17:00
🟥 Average change in ipInUnknownProtos per five seconds
RRDTOOL / TOBI OETIKER

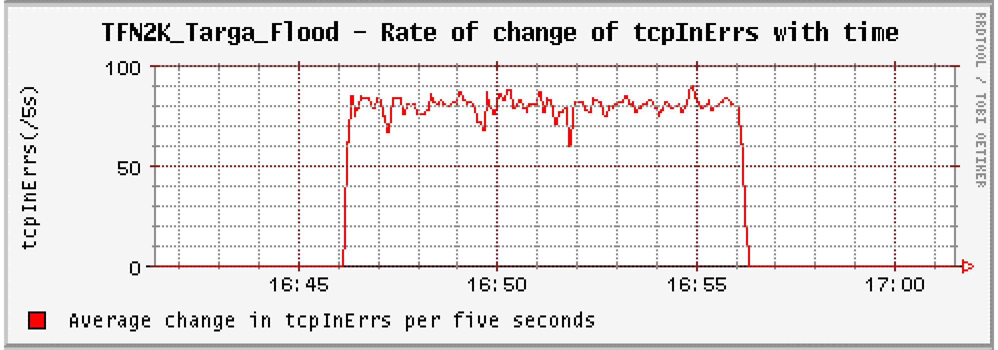

TFN2K_Targa_Flood - Rate of change of ipOutRequests with time
ipOutRequests(/5s)
600
400
200
0
16:45
16:50
16:55
17:00
🟥 Average change in ipOutRequests per five seconds
RRDTOOL / TOBI OETIKER



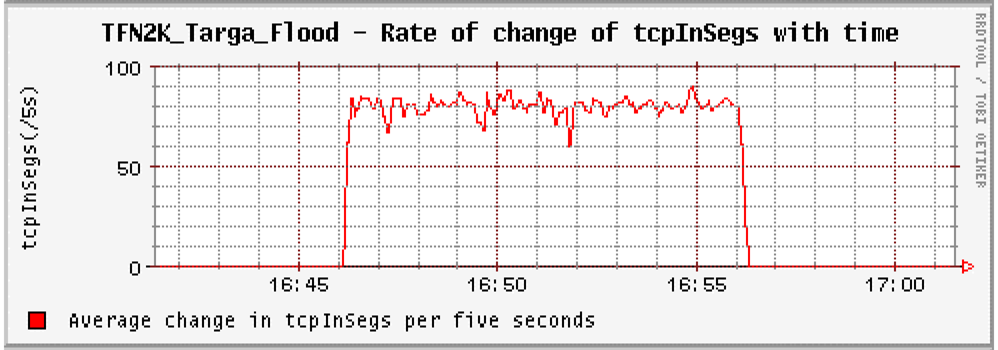
TFN2K_Targa_Flood - Rate of change of ipReasmFails with time
ipReasmFails(/5s)
1.0 k
0.0
16:45
16:50
16:55
17:00
🟥 Average change in ipReasmFails per five seconds
RRDTOOL / TOBI OETIKER



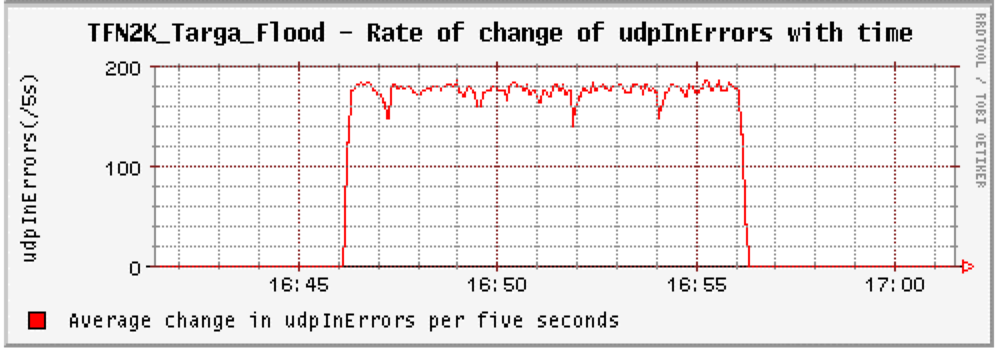
TFN2K_Targa_Flood - Rate of change of ipReasmReqds with time
ipReasmReqds(/5s)
1.0 k
0.0
16:45
16:50
16:55
17:00
🟥 Average change in ipReasmReqds per five seconds
RRDTOOL / TOBI OETIKER



TFN2K_Targa_Flood - Rate of change of tcpInErrs with time
tcpInErrs(/5s)
100
50
0
16:45
16:50
16:55
17:00
🟥 Average change in tcpInErrs per five seconds
RRDTOOL / TOBI OETIKER



TFN2K_Targa_Flood - Rate of change of tcpInSegs with time
tcpInSegs(/5s)
100
50
0
16:45
16:50
16:55
17:00
🟥 Average change in tcpInSegs per five seconds
RRDTOOL / TOBI OETIKER



TFN2K_Targa_Flood - Rate of change of udpInErrors with time
udpInErrors(/5s)
200
100
0
16:45
16:50
16:55
17:00
🟥 Average change in udpInErrors per five seconds
RRDTOOL / TOBI OETIKER

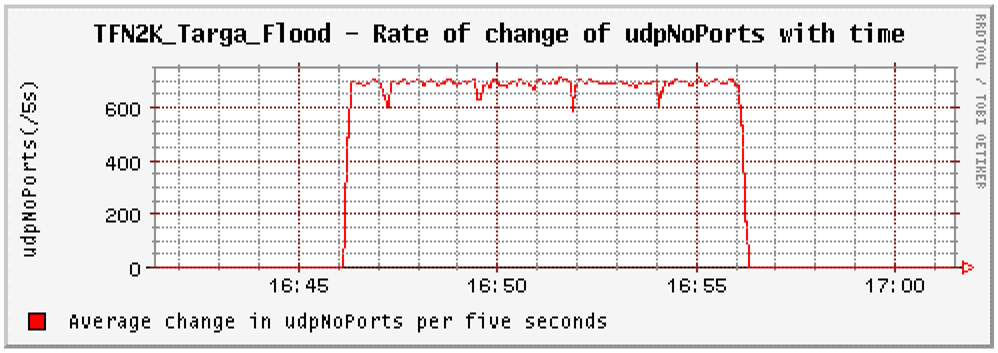
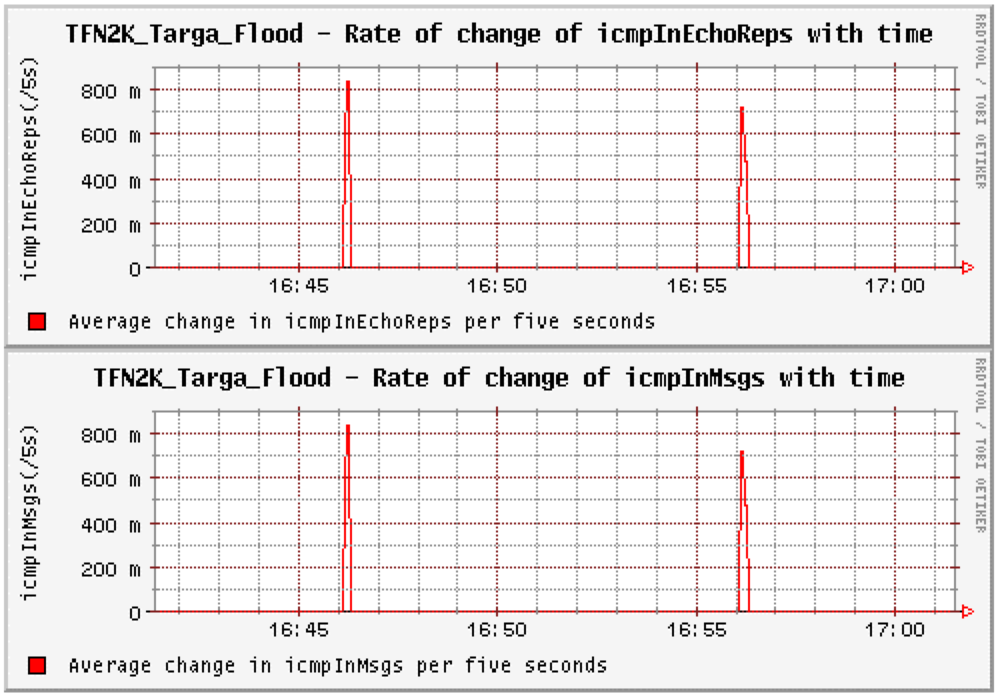
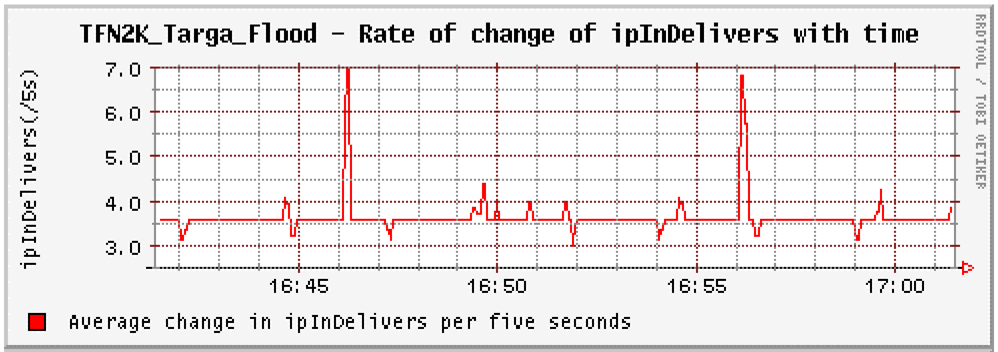


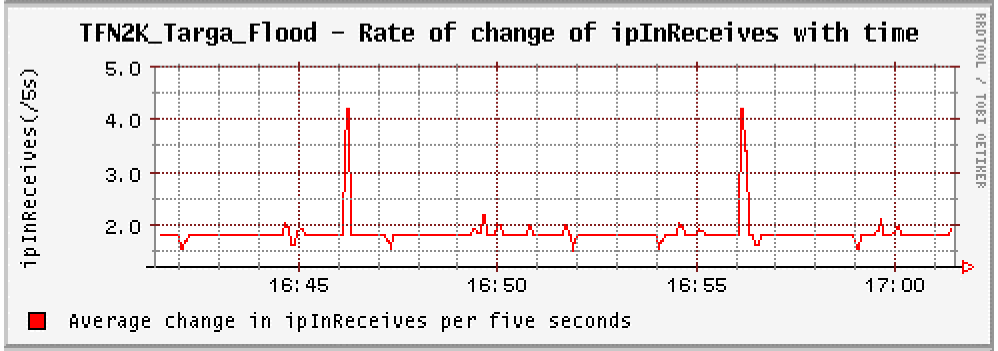
Figure 27. TFN2K Targa Flood Attack - Changes in Target MIB Variables.

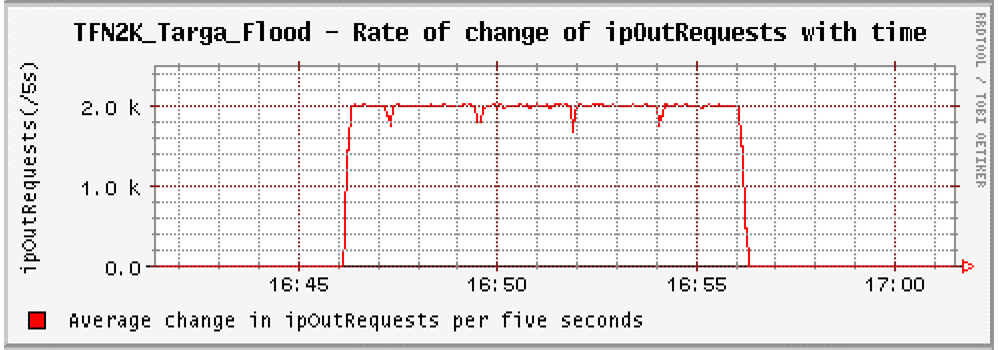
b. Changes in the Attacker-1 MIB Variables

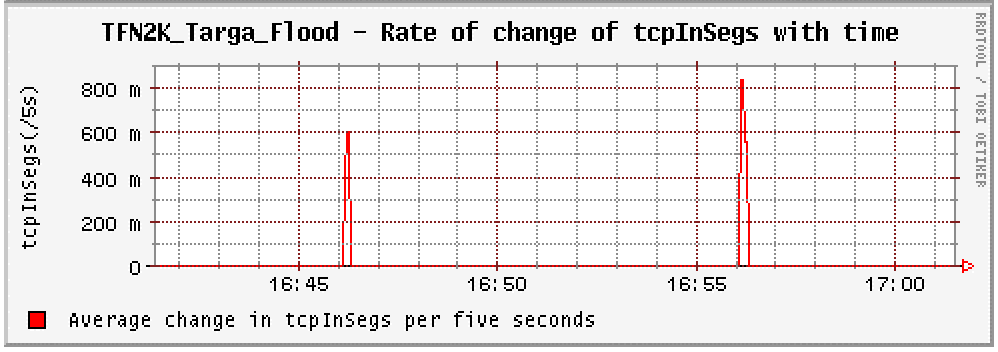
Figure 28 below shows the average change in the attacker1 MIB variables per five-second interval for a TFN2K targa-flood attack.




TFN2K_Targa_Flood - Rate of change of ipInDelivers with time
ipInDelivers(/5s)
7.0
6.0
5.0
4.0
3.0
16:45
16:50
16:55
17:00
RRDTOOL / TOBI OETIKER
Average change in ipInDelivers per five seconds


TFN2K_Targa_Flood - Rate of change of ipInReceives with time
ipInReceives(/5s)
5.0
4.0
3.0
2.0
16:45
16:50
16:55
17:00
RRDTOOL / TOBI OETIKER
Average change in ipInReceives per five seconds


TFN2K_Targa_Flood - Rate of change of ipOutRequests with time
ipOutRequests(/5s)
2.0 k
1.0 k
0.0
16:45
16:50
16:55
17:00
RRDTOOL / TOBI OETIKER
Average change in ipOutRequests per five seconds


TFN2K_Targa_Flood - Rate of change of tcpInSegs with time
tcpInSegs(/5s)
800 m
600 m
400 m
200 m
0
16:45
16:50
16:55
17:00
RRDTOOL / TOBI OETIKER
Average change in tcpInSegs per five seconds

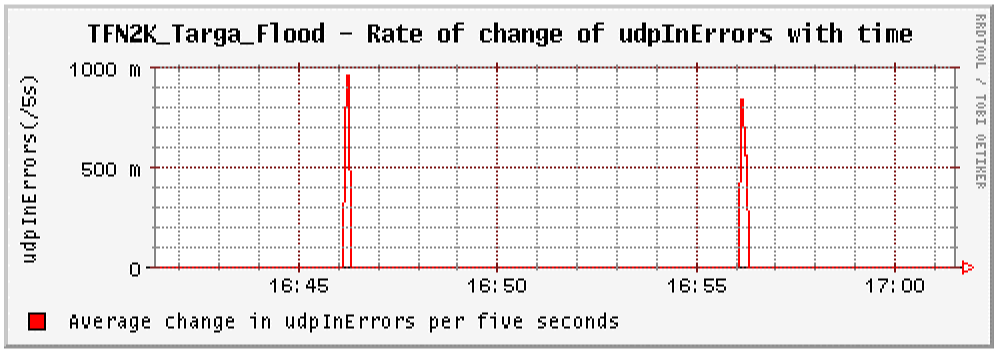
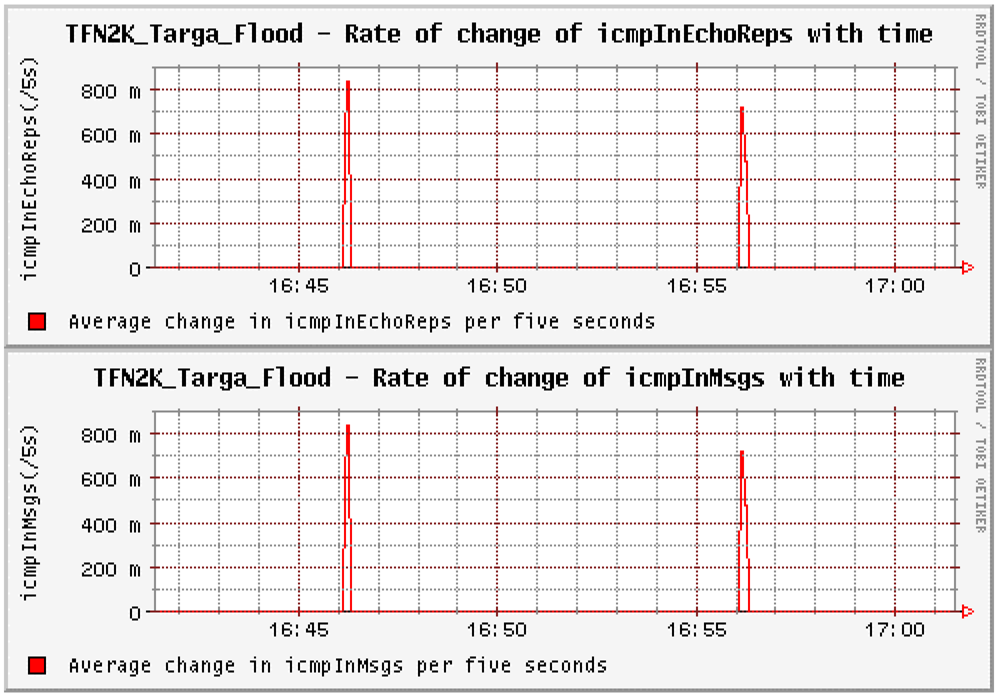
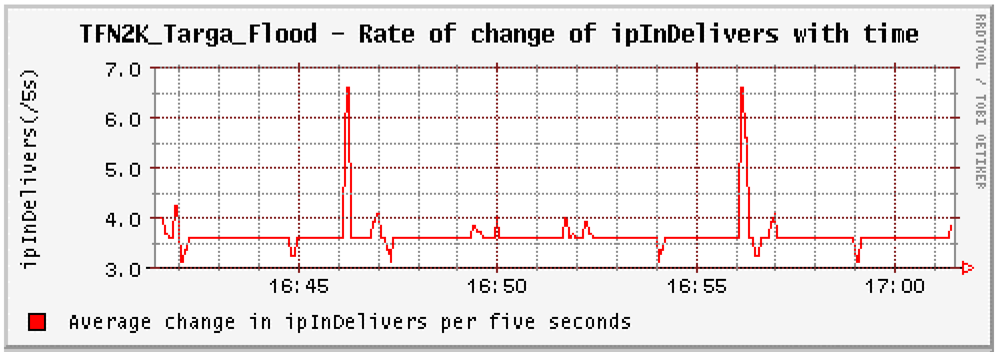


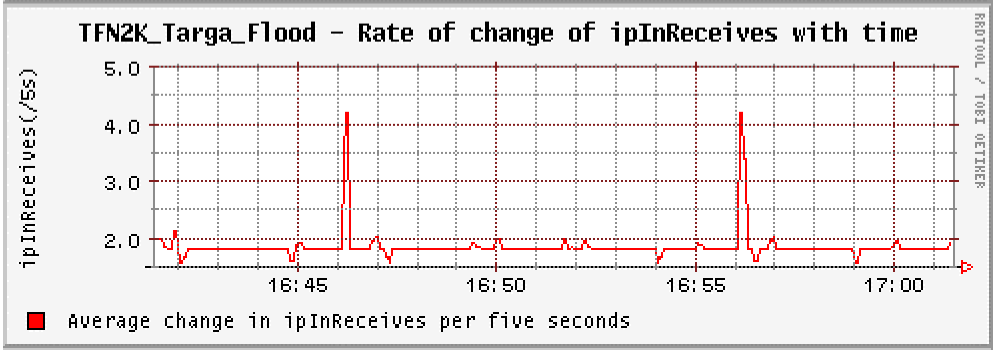
Figure 28. TFN2K Targa Flood Attack - Changes in Attacker1 MIB Variables.

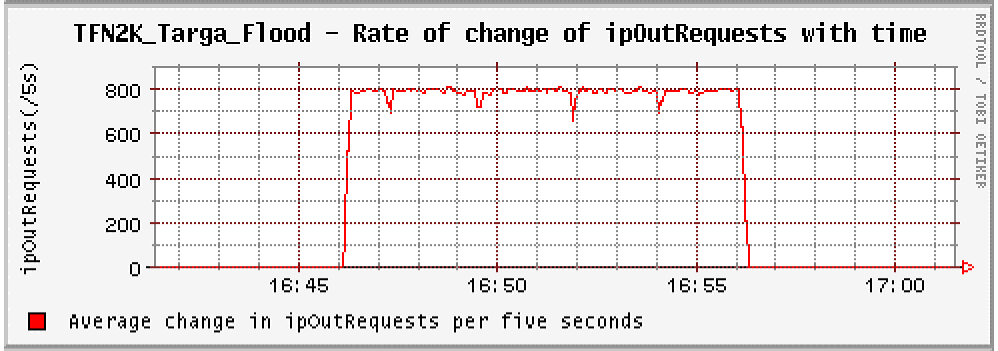
c. Changes in the Attacker-2 MIB Variables

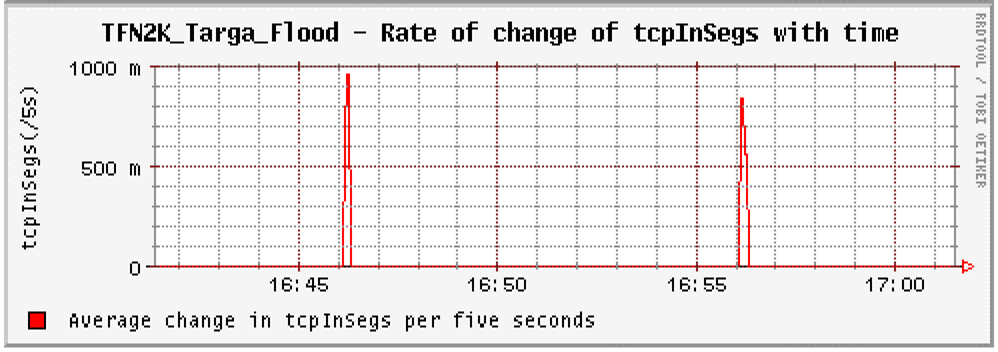
Figure 29 below shows the average change in the attacker2 MIB variables per five-second interval for a TFN2K targa-flood attack.




TFN2K_Targa_Flood - Rate of change of ipInDelivers with time
ipInDelivers(/5s)
7.0
6.0
5.0
4.0
3.0
16:45
16:50
16:55
17:00
RRDTOOL / TOBI OETIKER
🟥 Average change in ipInDelivers per five seconds


TFN2K_Targa_Flood - Rate of change of ipInReceives with time
ipInReceives(/5s)
5.0
4.0
3.0
2.0
16:45
16:50
16:55
17:00
RRDTOOL / TOBI OETIKER
🟥 Average change in ipInReceives per five seconds


TFN2K_Targa_Flood - Rate of change of ipOutRequests with time
ipOutRequests(/5s)
800
600
400
200
0
16:45
16:50
16:55
17:00
RRDTOOL / TOBI OETIKER
🟥 Average change in ipOutRequests per five seconds


TFN2K_Targa_Flood - Rate of change of tcpInSegs with time
tcpInSegs(/5s)
1000 m
500 m
0
16:45
16:50
16:55
17:00
RRDTOOL / TOBI OETIKER
🟥 Average change in tcpInSegs per five seconds

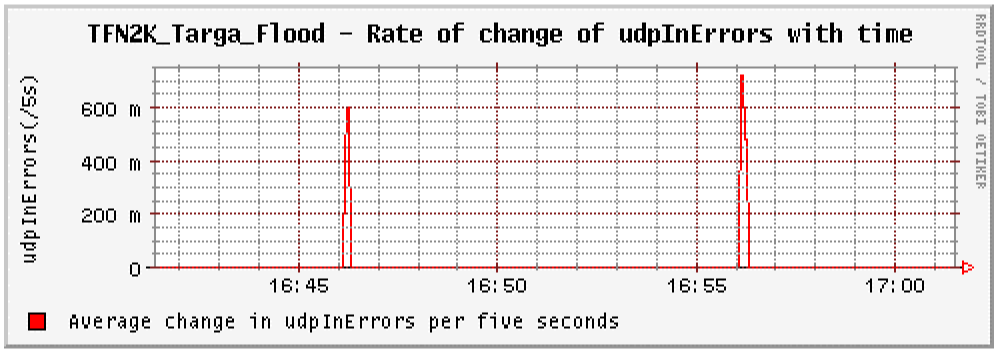


Figure 29. TFN2K Targa Flood Attack - Changes in Attacker2 MIB Vvariables.

5. Mix Flood

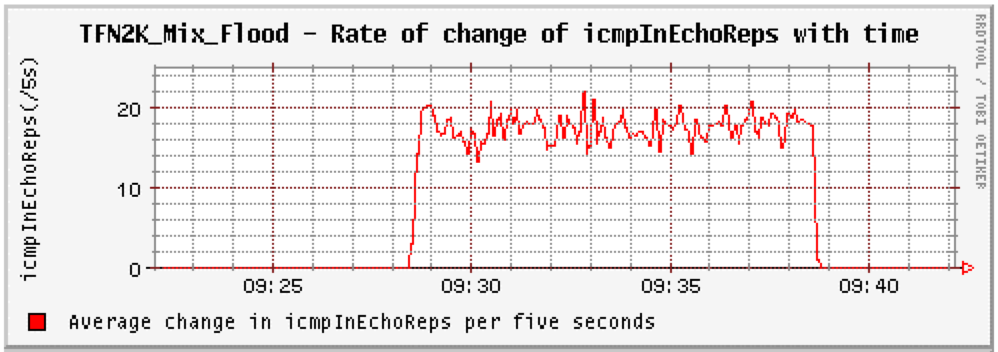
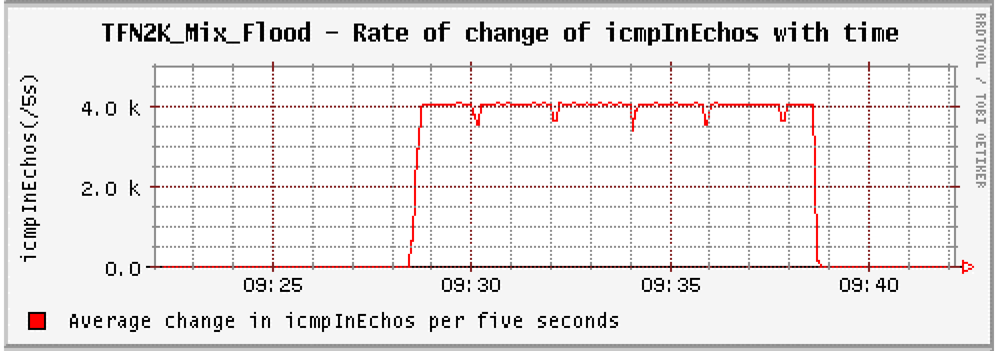
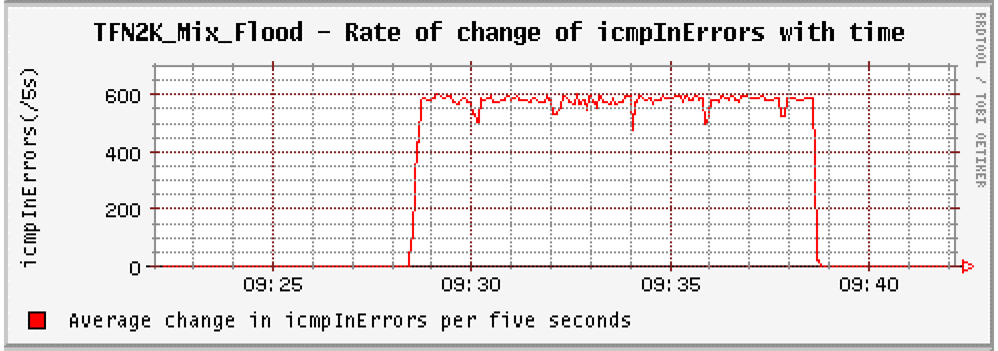
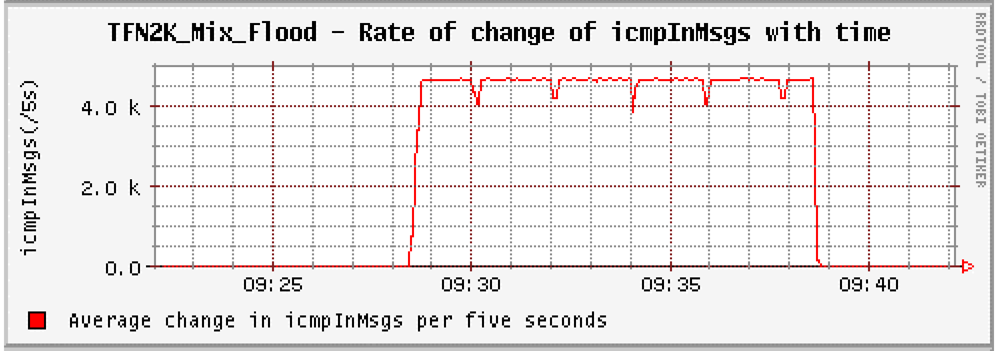
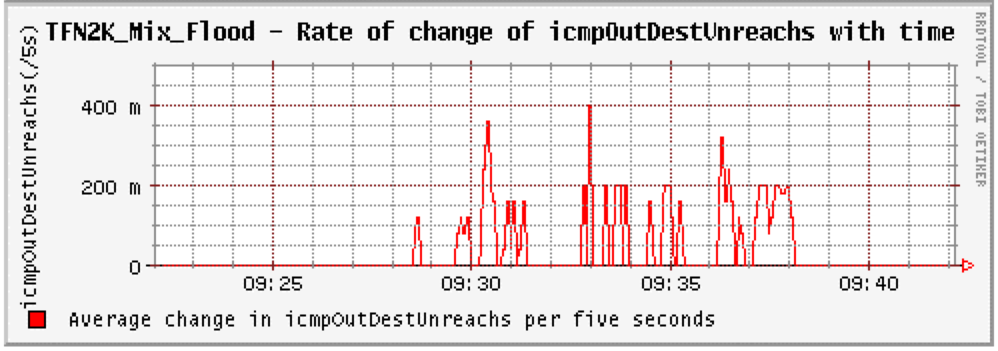
The affected variables at the target and at the attacker in a TFN2K mix-flood attack are shown in table 9.

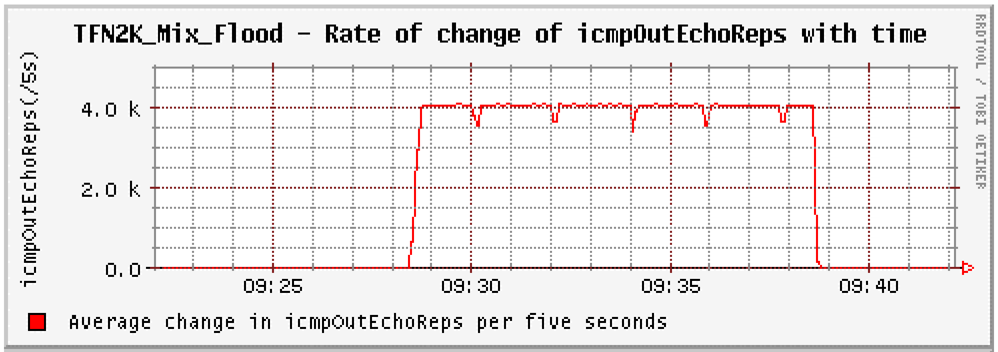
Sl. No.	Target	Attacker
1.	icmp.icmpInEchoReps.0	icmp.icmpInEchoReps.0
2.	**icmp.icmpInEchos.0**	icmp.icmpInMsgs.0
3.	**icmp.icmpInErrors.0**	ip.ipInDelivers.0
4.	**icmp.icmpInMsgs.0**	ip.ipInReceives.0
5.	**icmp.icmpOutDestUnreachs.0**	**ip.ipOutRequests.0**
6.	**icmp.icmpOutEchoReps.0**	tcp.tcpInSegs.0
7.	**icmp.icmpOutMsgs.0**	udp.udpInErrors.0
8.	**ip.ipInDelivers.0**	
9.	**ip.ipInReceives.0**	
10.	**ip.ipOutRequests.0**	
11.	**tcp.tcpInErrs.0**	
12.	**tcp.tcpInSegs.0**	
13.	**tcp.tcpOutRsts.0**	
14.	**tcp.tcpOutSegs.0**	
15.	udp.udpNoPorts.0	
16.	**udp.udpInErrors.0**	

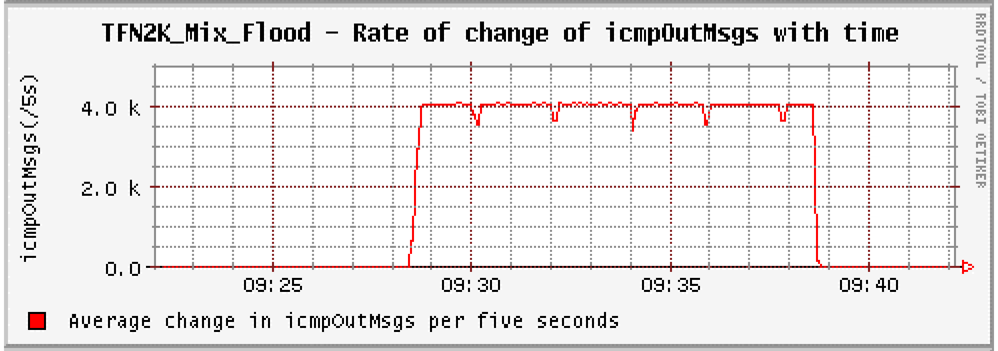
Table 9. Affected MIB Variables In TFN2K Mix Flood Attack.

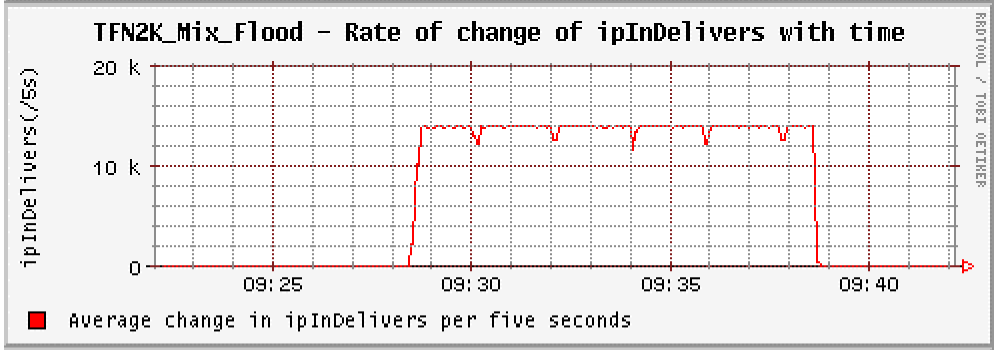
a. Changes in the Target MIB Variables

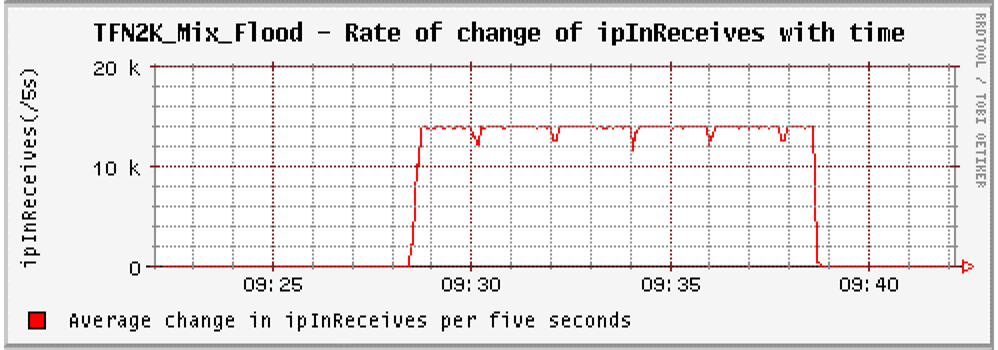
Figure 30 below shows the average change in the target MIB variables per five-second interval for a TFN2K mix-flood attack.

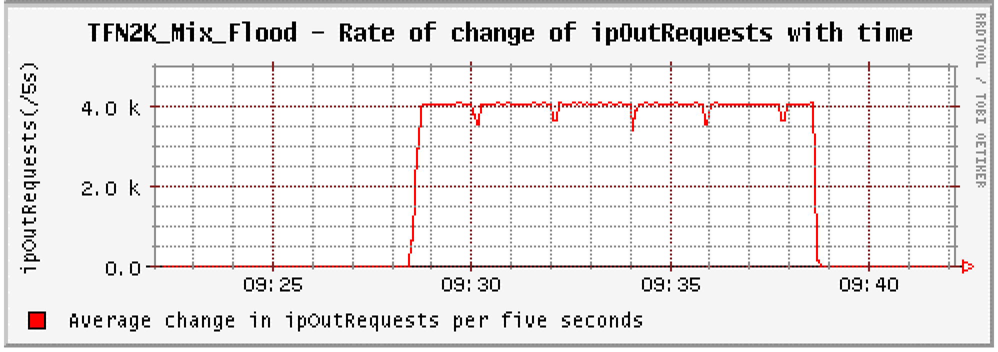

TFN2K_Mix_Flood - Rate of change of icmpInEchoReps with time
icmpInEchoReps(/5s)
20
10
0
09:25
09:30
09:35
09:40
Average change in icmpInEchoReps per five seconds
RRDTOOL / TOBI OETIKER

TFN2K_Mix_Flood - Rate of change of icmpInEchos with time
icmpInEchos(/5s)
4.0 k
2.0 k
0.0
09:25
09:30
09:35
09:40
Average change in icmpInEchos per five seconds
RRDTOOL / TOBI OETIKER

TFN2K_Mix_Flood - Rate of change of icmpInErrors with time
icmpInErrors(/5s)
600
400
200
0
09:25
09:30
09:35
09:40
Average change in icmpInErrors per five seconds
RRDTOOL / TOBI OETIKER

TFN2K_Mix_Flood - Rate of change of icmpInMsgs with time
icmpInMsgs(/5s)
4.0 k
2.0 k
0.0
09:25
09:30
09:35
09:40
Average change in icmpInMsgs per five seconds
RRDTOOL / TOBI OETIKER

TFN2K_Mix_Flood - Rate of change of icmpOutDestUnreachs with time
icmpOutDestUnreachs(/5s)
400 m
200 m
0
09:25
09:30
09:35
09:40
Average change in icmpOutDestUnreachs per five seconds
RRDTOOL / TOBI OETIKER

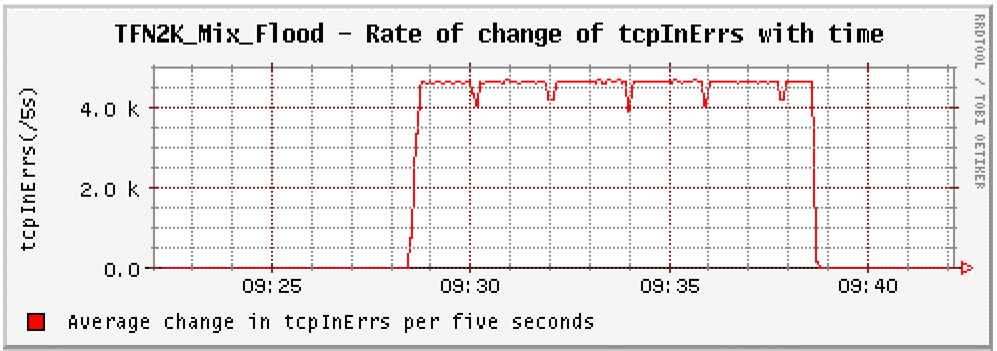

TFN2K_Mix_Flood - Rate of change of icmpOutEchoReps with time
icmpOutEchoReps(/5s)
4.0 k
2.0 k
0.0
09:25
09:30
09:35
09:40
RRDTOOL / TOBI OETIKER
🟥 Average change in icmpOutEchoReps per five seconds

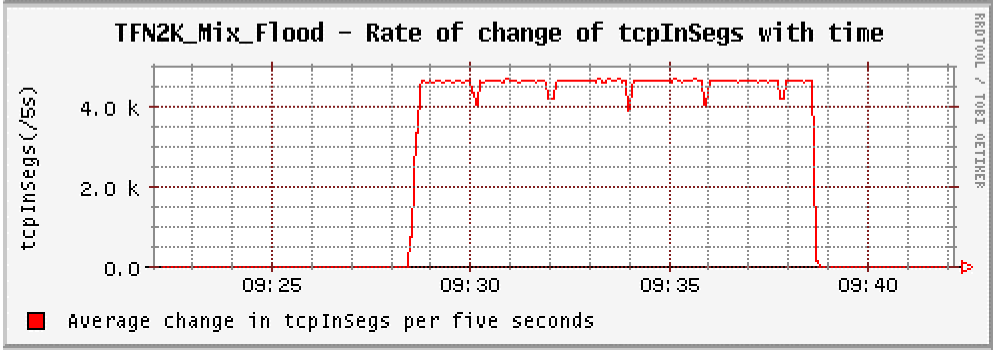

TFN2K_Mix_Flood - Rate of change of icmpOutMsgs with time
icmpOutMsgs(/5s)
4.0 k
2.0 k
0.0
09:25
09:30
09:35
09:40
RRDTOOL / TOBI OETIKER
🟥 Average change in icmpOutMsgs per five seconds

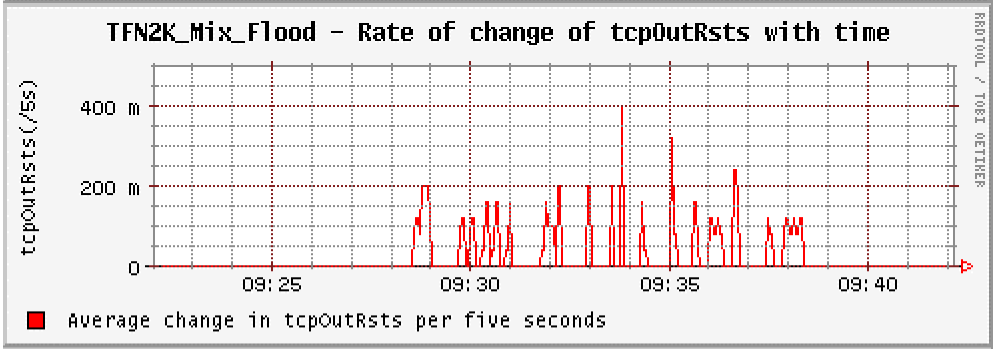

TFN2K_Mix_Flood - Rate of change of ipInDelivers with time
ipInDelivers(/5s)
20 k
10 k
0
09:25
09:30
09:35
09:40
RRDTOOL / TOBI OETIKER
🟥 Average change in ipInDelivers per five seconds

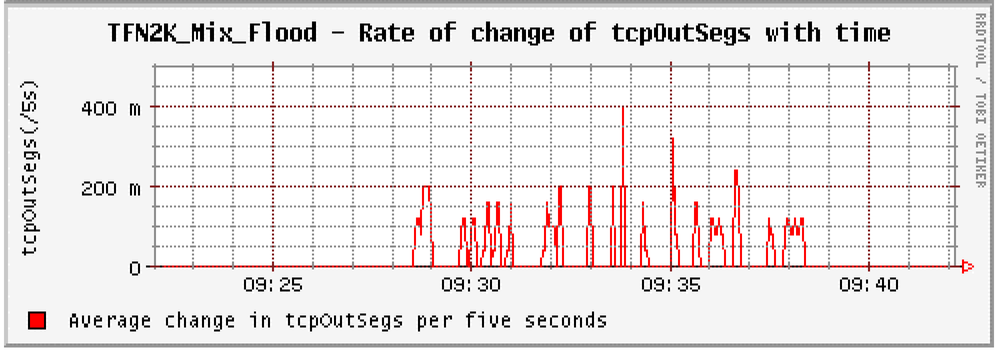

TFN2K_Mix_Flood - Rate of change of ipInReceives with time
ipInReceives(/5s)
20 k
10 k
0
09:25
09:30
09:35
09:40
RRDTOOL / TOBI OETIKER
🟥 Average change in ipInReceives per five seconds

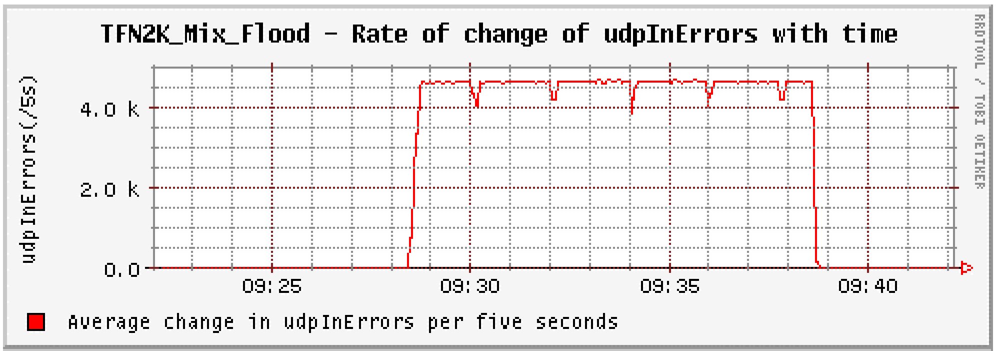

TFN2K_Mix_Flood - Rate of change of ipOutRequests with time
ipOutRequests(/5s)
4.0 k
2.0 k
0.0
09:25
09:30
09:35
09:40
RRDTOOL / TOBI OETIKER
🟥 Average change in ipOutRequests per five seconds


TFN2K_Mix_Flood - Rate of change of tcpInErrs with time
tcpInErrs(/5s)
4.0 k
2.0 k
0.0
09:25
09:30
09:35
09:40
🟥 Average change in tcpInErrs per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_Mix_Flood - Rate of change of tcpInSegs with time
tcpInSegs(/5s)
4.0 k
2.0 k
0.0
09:25
09:30
09:35
09:40
🟥 Average change in tcpInSegs per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_Mix_Flood - Rate of change of tcpOutRsts with time
tcpOutRsts(/5s)
400 m
200 m
0
09:25
09:30
09:35
09:40
🟥 Average change in tcpOutRsts per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_Mix_Flood - Rate of change of tcpOutSegs with time
tcpOutSegs(/5s)
400 m
200 m
0
09:25
09:30
09:35
09:40
🟥 Average change in tcpOutSegs per five seconds
RRDTOOL / TOBI OETIKER



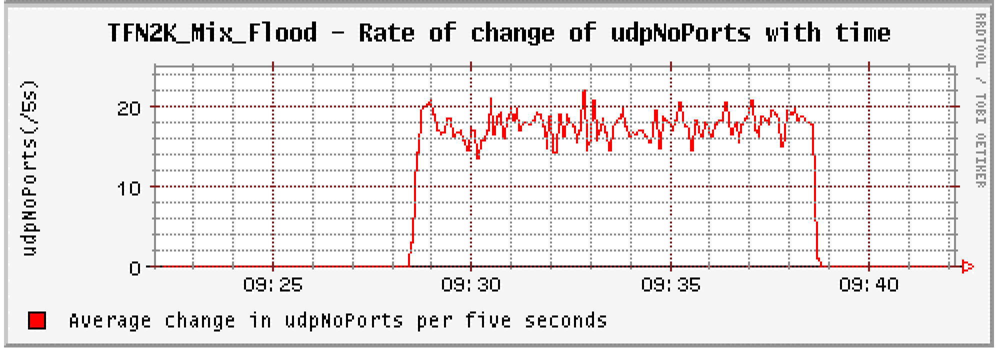
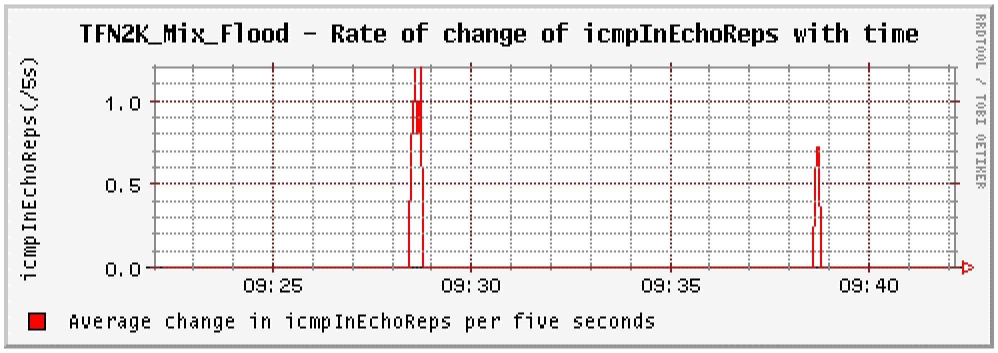
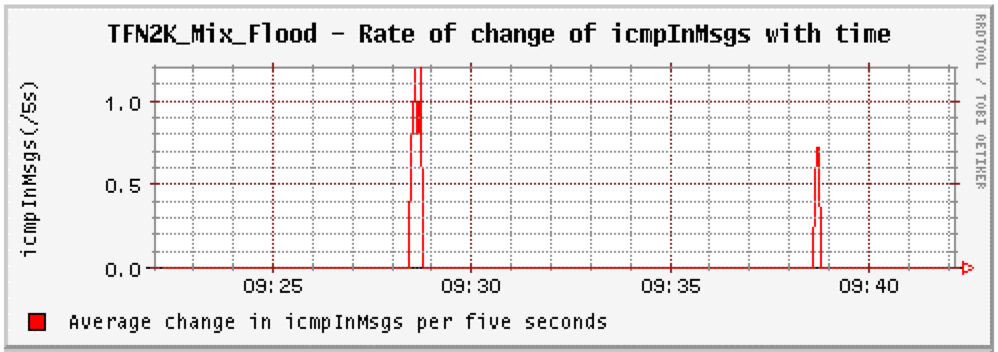


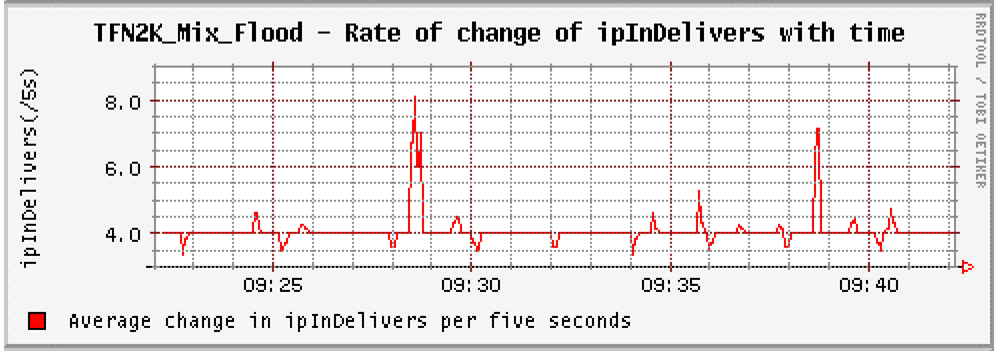
Figure 30. TFN2K Mix Flood Attack - Changes in Target MIB Variables.

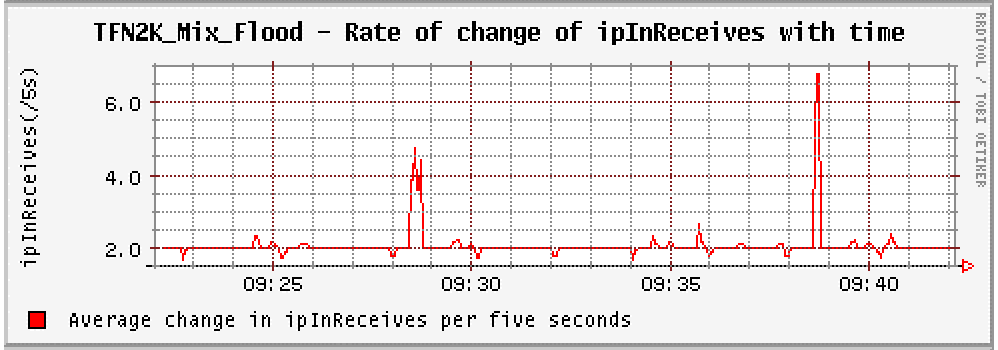
b. Changes in the Attacker-1 MIB Variables

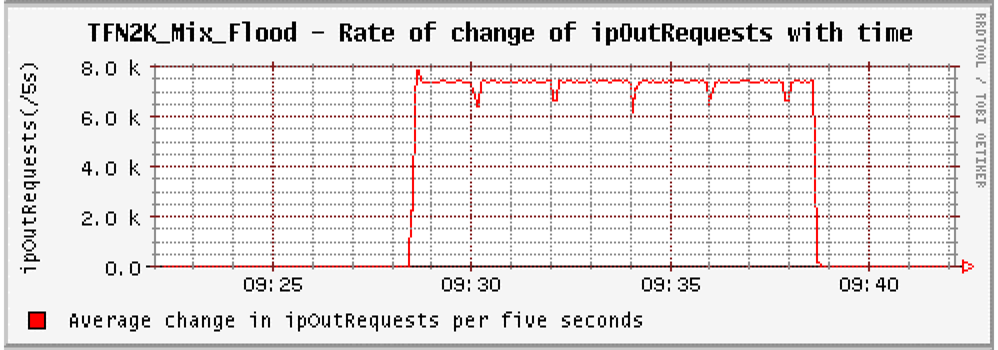
Figure 31 below shows the average change in the attacker1 MIB variables per five-second interval for a TFN2K mix-flood attack.

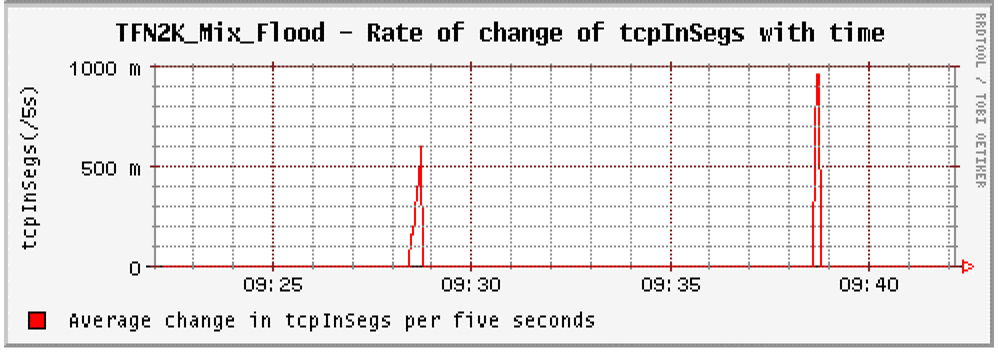



TFN2K_Mix_Flood - Rate of change of icmpInMsgs with time
icmpInMsgs(/5s)
1.0
0.5
0.0
09:25
09:30
09:35
09:40
🟥 Average change in icmpInMsgs per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_Mix_Flood - Rate of change of ipInDelivers with time
ipInDelivers(/5s)
8.0
6.0
4.0
09:25
09:30
09:35
09:40
🟥 Average change in ipInDelivers per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_Mix_Flood - Rate of change of ipInReceives with time
ipInReceives(/5s)
6.0
4.0
2.0
09:25
09:30
09:35
09:40
🟥 Average change in ipInReceives per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_Mix_Flood - Rate of change of ipOutRequests with time
ipOutRequests(/5s)
8.0 k
6.0 k
4.0 k
2.0 k
0.0
09:25
09:30
09:35
09:40
🟥 Average change in ipOutRequests per five seconds
RRDTOOL / TOBI OETIKER


TFN2K_Mix_Flood - Rate of change of tcpInSegs with time
tcpInSegs(/5s)
1000 m
500 m
0
09:25
09:30
09:35
09:40
🟥 Average change in tcpInSegs per five seconds
RRDTOOL / TOBI OETIKER

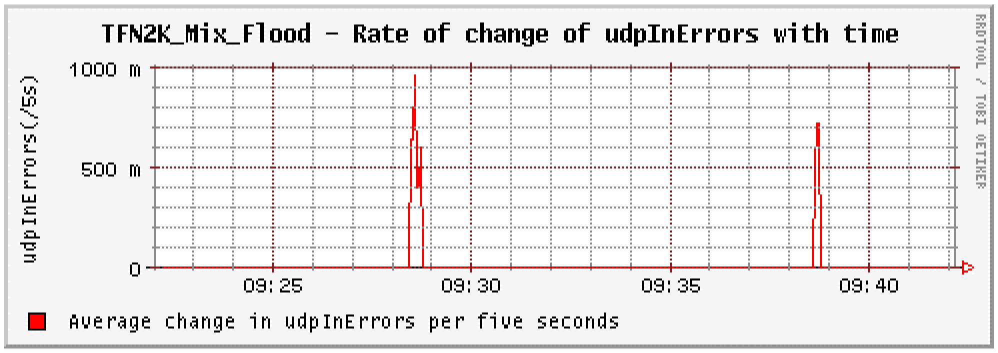
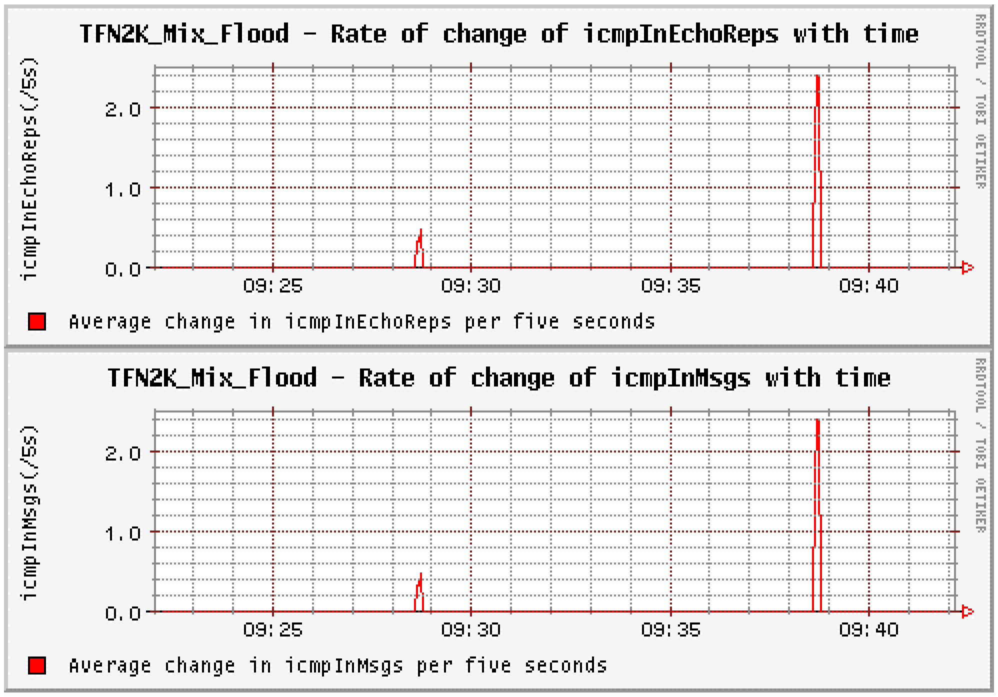
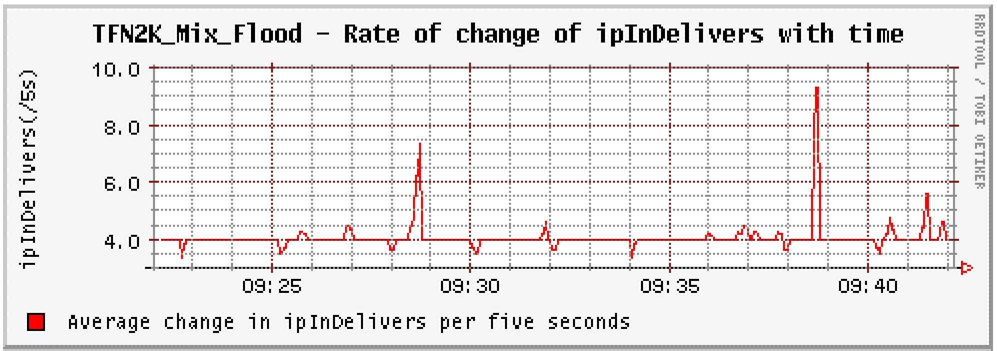


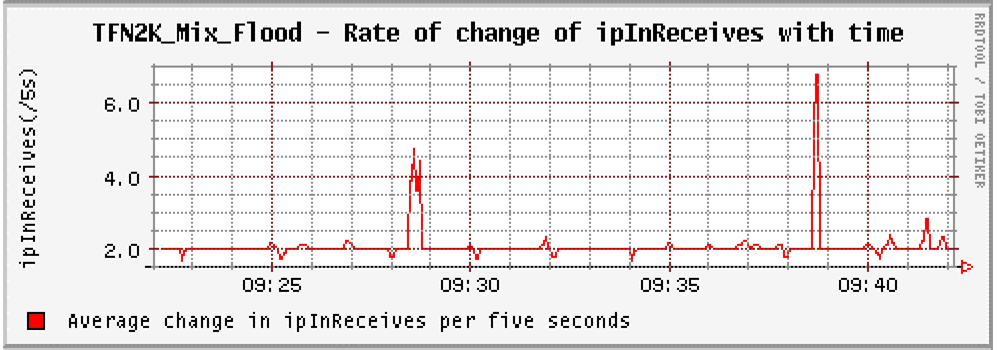
Figure 31. TFN2K Mix Flood Attack - Changes in Attacker1 MIB Variables.

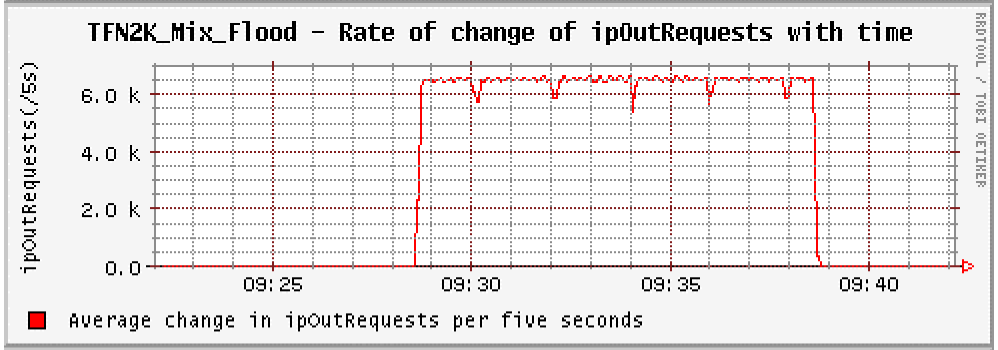
c. Changes in the Attacker-2 MIB Variables

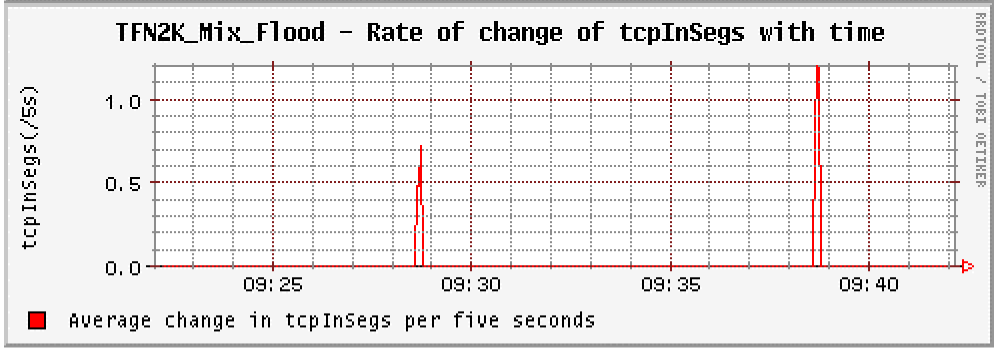
Figure 32 below shows the average change in the attacker2 MIB variables per five-second interval for a TFN2K mix-flood attack.




TFN2K_Mix_Flood - Rate of change of ipInDelivers with time
ipInDelivers(/5s)
10.0
8.0
6.0
4.0
09:25
09:30
09:35
09:40
RRDTOOL / TOBI OETIKER
🟥 Average change in ipInDelivers per five seconds


TFN2K_Mix_Flood - Rate of change of ipInReceives with time
ipInReceives(/5s)
6.0
4.0
2.0
09:25
09:30
09:35
09:40
RRDTOOL / TOBI OETIKER
🟥 Average change in ipInReceives per five seconds


TFN2K_Mix_Flood - Rate of change of ipOutRequests with time
ipOutRequests(/5s)
6.0 k
4.0 k
2.0 k
0.0
09:25
09:30
09:35
09:40
RRDTOOL / TOBI OETIKER
🟥 Average change in ipOutRequests per five seconds


TFN2K_Mix_Flood - Rate of change of tcpInSegs with time
tcpInSegs(/5s)
1.0
0.5
0.0
09:25
09:30
09:35
09:40
RRDTOOL / TOBI OETIKER
🟥 Average change in tcpInSegs per five seconds

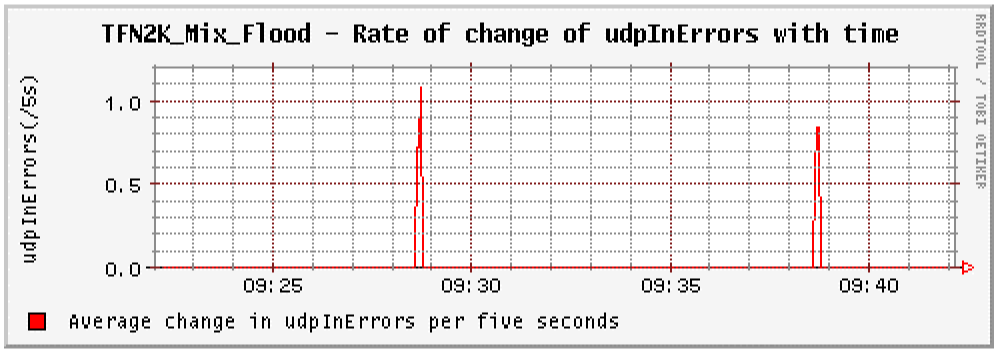


Figure 32. TFN2K Mix Flood Attack - Changes in Attacker2 MIB Variables.

D. SUMMARY

The summary of the various MIB variables affected, is given in the Table 10 for the attackers and in Table 11 for the targets for different attacks.

Sl	**MIB**	**Trinoo**	**TFN**			**TFN2K**				
			Ping	**Syn**	**UDP**	**Ping**	**Syn**	**UDP**	**Targa**	**Mix**
	ICMP									
1.	icmpInDestUnreachs				XX				XX	
2.	icmpInEchoReps		X			X				X
3.	icmpInEchos		XX			XX				XX
4.	icmpInErrors		X			XX			XX	XX
5.	icmpInMsgs		XX		XX	XX			XX	XX
6.	icmpOutDestUnreachs	XX			XX			X	XX	XX
7.	icmpOutEchoReps		XX			XX				XX
8.	icmpOutMsgs	XX	XX		XX	XX		X	XX	XX
	IP									
1.	ipInDelivers	XX	XX	XX	XX	XX	XX	XX	XX	XX
2.	ipInReceives	XX	XX	XX	XX	XX	XX	XX	XX	XX
3.	ipInUnknownProtos								XX	
4.	ipOutRequests	XX	XX		XX	XX	X	X	XX	XX
5.	ipReasmFails								XX	
6.	ipReasmReqds								XX	
	TCP									
1.	tcpInErrs			XX			XX		X	XX
2.	tcpInSegs			XX			XX		X	XX

3.	tcpOutRsts						XX			XX
4.	tcpOutSegs						XX			XX
	UDP									
1.	udpInDatagrams	X								
2.	udpInErrors				XX			XX	X	XX
3.	udpNoPorts	XX	X		XX	XX		X	XX	X
4.	udpOutDatagrams	X							X	

X – Small Variation XX – Large Variation

Table 10. Summary Of Affected MIB Variables In Targets In DDoS Attacks.

Sl No.	**MIB**	**Tri noo**	**TFN**			**TFN2K**				
			Ping	**Syn**	**UDP**	**Ping**	**Syn**	**UDP**	**Targa**	**Mix**
	ICMP									
1.	icmpInDestUnreachs	XX								
2.	icmpInEchoReps		X	X	X	X	X	X	X	X
3.	icmpInEchos									
4.	icmpInErrors									
5.	icmpInMsgs	XX	X	X	X	X	X	X	X	X
6.	icmpOutDestUnreachs									
7.	icmpOutEchoReps									
8.	icmpOutMsgs									
	IP									
1.	ipInDelivers	X				X	X	X	X	X
2.	ipInReceives	XX				X		X	X	X
3.	ipInUnknownProtos									
4.	ipOutRequests	XX	XX	XX	XX	XX	XX	XX	XX	XX
5.	ipReasmFails									
6.	ipReasmReqds									
	TCP									
1.	tcpInErrs									
2.	tcpInSegs					X	XX	X	X	X
3.	tcpOutRsts									

4.	tcpOutSegs									
	UDP									
1.	udpInDatagrams	X								
2.	udpInErrors					X	X	X	X	X
3.	udpNoPorts									
4.	udpOutDatagrams	XX							X	

X – Small Variation XX – Large Variation

Table 11. Summary Of Affected MIB Variables In Attackers In DDoS Attacks.

The analyses of the result obtained shows that a large increase in the ip.ipInReceives.0 and ip.ipInDelivers.0 MIBs on a particular network device is a sure indication of it being attacked in a DDoS manner. On the other hand a large raise in the ip.ipOutRequests.0 MIB variable means that the system has been compromised and is a zombie participating in a DDoS/DoS attack against some other system. The monitoring system can further home on to the exact type of attack based on the signatures of the attacks as seen from the tables above.

Another, important property as noticed from the graphs is that before the attacker commences the attack there are some minor changes in its MIB values. A mathematical correlation between these values and the attack carried out has been shown mathematically [JOAO-01]. This knowledge could be used to detect the DDoS attack proactively.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

This thesis explored the usage of one commercial NMS (Spectrum) for observing the DDoS attacks. However this NMS can poll the hosts only at an interval greater than or equal to 30 seconds. Hence, it was found to be unsuitable for carrying out the experiments. The program using ucd-snmp library was used thereafter to log the variations in the various MIB variables and the results showed that there is indeed a large and sudden variation in the various SNMP MIB variables. These could be monitored by a NMS provided the sampling/querying time for the NMS is short enough to quickly detect the peak pattern usage.

Even though a large number of MIB variables are affected in a DDoS attack this thesis identified the MIB variables of importance (Key variables). These key variables could be grouped together and monitored by the NMS for monitoring purpose so as to be better able to carry out its Security Management tasks. Continuous monitoring of only these MIBs by a NMS is required for achieving the desired result.

The thesis determined the key variables for not only the attacked systems but also the attacker systems. Thus, a possibility exists for using NMS for preventing the hosts within the network from taking part in attack against any external host, but the results clearly show that most of the variables on the attackers have a very small change for a short period of time and so the NMS will have to query the hosts at a shorter duration of time. However, this shorter polling time will result in increase in the traffic in the internal network and so a better alternative is to have intelligent SNMP agents which monitor the normal variable profile and on observing any deviation raise a trap for the NMS. A large number of such traps from a number of hosts within the network will be a sure indicator of compromised hosts within the network.

B. RECOMMENDATIONS

Many areas remain for further research. Even though this thesis provided ample evidence of the variation in the MIB traffic, which could be used by the Network Management System, the actual automation of the NMS with the observed changes could be an area of active research.

Another active area of research could be to actually determine the reactive action which could be taken to prevent the Distributed Denial of Service attacks or at the very least to minimize it. The NMS could trigger the firewall or any packet filter application to stop these renegade packets from entering its network.

Going further ahead, as mentioned earlier, the SNMP agents in the host systems could be modified to have some sort of intelligence to continuously monitor these key variables and flag an alarm in case of any observed variation in their usage pattern. The alarm could be flagged by means of raising a trap. A study could also be made to determine if this is a more effective approach than current IDS technology.

APPENDIX PROGRAMS

A. MIBLOGGER.C

```
/*
 *      Program Name:       miblogger.c
 *      Author:             Chandan Singh Negi
 *      Description:        A simple program to query a list of hosts for the value of
 *                          mib variables at regular time intervals
 *      Usage:              Compile and run without any command line parameters.
 *                          You will have to modify the mib variables and the host
 *                          addresses in the code.
 *      Platform:           RedHat Linux 7.0 – kernel 2.2.16-22
 *      Library:            ucd-snmp
 *      Compiler:           gcc –v 2.96
 */


#include <ucd-snmp/ucd-snmp-config.h>
#include <ucd-snmp/ucd-snmp-includes.h>
#include <sys/time.h>                    // for finding the current time

#define POLL_INTERVAL  5                 // our query interval


/*
 *   A list of hosts and host-type to query
 */

struct host {
        char *name;
        char *type;
} hosts[] = {
        {"209.60.77.20","ATTACKER"},
        { "209.60.77.24", "ATTACKER" },
        { "209.60.77.23", "TARGET"  },
        {NULL}
};


/*
 *      A list of mib-variables to query the attacker and targets
 */

struct oid {
        char *Name;
        oid Oid[MAX_OID_LEN];
        int OidLen;
} oids_T[] = {
        { "udp.udpInDatagrams.0" },
        { "udp.udpOutDatagrams.0" },
        { "udp.udpNoPorts.0" },
        { "ip.ipInReceives.0" },
        { "ip.ipInDelivers.0" },
        { "ip.ipOutRequests.0"},
        { "icmpOutMsgs.0" },
        { "icmpOutDestUnreachs.0" },
        {NULL}
};

struct oid oids_A[] = {
        { "udp.udpInDatagrams.0" },
        { "udp.udpOutDatagrams.0" },
        { "ip.ipInReceives.0" },
        { "ip.ipInDelivers.0" },
        { "ip.ipOutRequests.0"},
        { "icmpInMsgs.0"},
        { "icmpInDestUnreachs.0" },
        { NULL}
};


/*********************************************************************/

/*
 *      function:               void initialize(void)
 *      parameters:             none
 *      returns:                none
 *      description:            Reads the various mib variables and correlates them
 *                              with their oid number
 */

void initialize (void)
{
        int i;
        struct oid *op;
        init_snmp("miblogger");

        for(i=0; i<2; i++) {
                switch(i) {
                        case 0: op = oids_T;
```

```
                                break;
                        case 1: op = oids_A;
                } // end switch
                while (op->Name) {
                        op->OidLen = sizeof(op->Oid)/sizeof(op->Oid[0]);
                if (!read_objid(op->Name, op->Oid, &op->OidLen)) {
                        snmp_perror("read_objid");
                        exit(1);
                        } // end if
                op++;
                } // end while
        } //  end for
} // end initialize


/*********************************************************************/

/*
 *      function:               int log_response (int status, struct snmp_session *sp,
 *                                                      struct snmp_pdu *pdu, FILE *f)
 *      parameters:             status - the status returned on querying the hosts.
 *                              *sp -
 *                              *pdu - pointer to the PDU returned by the host
 *                              *f - the file where the response is to be logged
 *      returns:                0 - if the response PDU from the queried host contained
 *                                  error
 *                              1 - if no error, if there is timeout while querying host,
 *                                   or if no specified host exists
 *      description:            logs the response returned by various hosts in their
 *                              respective log files
 *
 */

int log_response (int status, struct snmp_session *sp, struct snmp_pdu *pdu, FILE *f)
{
        char buf[128];
        struct variable_list *vp;
        int ix;
        time_t now;

        now = time(NULL);

        fprintf(stdout, "Logging the results\n");
        fprintf(stdout, "%u:",now);
      fprintf(f, "%u:",now);
```

```
        switch (status) {
        case STAT_SUCCESS:
                vp = pdu->variables;
                if (pdu->errstat == SNMP_ERR_NOERROR) {
                        while (vp) {
                        sprint_variable(buf, vp->name, vp->name_length, vp);
                        fprintf(stdout, "%s: %s\n", sp->peername, buf);
                        fprintf(f, "%s: %s\n", sp->peername, buf);
                        vp = vp->next_variable;
                        } //while
                } //if
                else {
    for (ix = 1; vp && ix != pdu->errindex; vp = vp->next_variable,  ix++);


                        if (vp) sprint_objid(buf, vp->name, vp->name_length);
                        else strcpy(buf, "(none)");
                        fprintf(stdout, "%s: %s: %s\n",
                                sp->peername, buf, snmp_errstring(pdu->errstat));
                        fprintf(f, "%s: %s: %s\n",
                                sp->peername, buf, snmp_errstring(pdu->errstat));
                } //else
                return 1;
        case STAT_TIMEOUT:
                fprintf(stdout, "%s: Timeout\n", sp->peername);
                return 0;
        case STAT_ERROR:
                snmp_perror(sp->peername);
                return 0;
        } //switch
  return 0;
}


/*****************************************************************/

/*
 *      function:               void print_new_line( FILE *f)
 *      parameters:             FILE *f - the file in which a new line is to be added
 *      returns:                none
 *      description:            adds a new line to the specified log file. This is being used
 *                              so as to make the parsing of the log files easier.
 */

void print_new_line( FILE *f)
{
        fprintf(f,"\n");
```

```
}


/**********************************************************************/

/*
 *      function:               void poll(void)
 *      parameters:             none
 *      returns:                none
 *      description:            polls the various hosts.
 */

void poll(void)
{
        struct host *hp;
        struct snmp_session ss, *sp;
        struct oid *op;

        FILE *f;
        int i;

        for ( i = 0; i<3; i++) {
                hp = &hosts[i];
                //f = fp[i];
                if (i==0) f = fopen("a1.log","a");
                else if (i==1) f = fopen("a2.log","a");
                else f = fopen("t.log","a");

                snmp_sess_init(&ss);                    // initialize session
                ss.version = SNMP_VERSION_2c;
                ss.peername = hp->name;
                ss.community = "public";
                ss.community_len = strlen(ss.community);
                snmp_synch_setup(&ss);
                if (!(sp = snmp_open(&ss))) {
                        snmp_perror("snmp_open");
                        continue;
                } //if

                print_new_line(f);

                if (hp->type == "ATTACKER") op = oids_A;
                else op = oids_T;
                for (op; op->Name; op++) {
                        struct snmp_pdu *req, *resp;
                        int status;
                        req = snmp_pdu_create(SNMP_MSG_GET);
                        snmp_add_null_var(req, op->Oid, op->OidLen);
                        status = snmp_synch_response(sp, req, &resp);
                        if (!log_response(status, sp, resp,f)) break;
                        snmp_free_pdu(resp);
                } //for
   snmp_close(sp);
        } //for
}


/*********************************************************************/

/*
 *      function:               int main (int argc, char *argv[])
 *      parameters:             argc - no of arguments
 *                              argv[] - an array of pointers to the command line
 *                              parameters
 *      returns:                0 - if executes successfully
 *                              1 -     if execution unsuccessful
 *      description:            the main function. Doesn't use any command line
 *                              parameters.
 */

int main (int argc, char *argv[])
{
        int i;
        initialize();

        for ( i=1; i<240; i++) {
        poll();
        sleep(POLL_INTERVAL);
        }
        return 0;
}
```

B. REVERSE.PL

```
#!      perl –w
#       Author:         Chandan Singh Negi
#       Description:    A short script to reverse the log file to increasing order of time
#       Usage:          perl –w reverse.pl logfilename

my (@array);
my ($x,$y) = 0;
```

```
open(IN, "<".$ARGV[0]) || die "Can't open input file\n";
open(OUT, ">". $ARGV[1]) || die "Can't open output file\n";

while(<IN>) {
        $array[$x++]= $_;
}

close(IN);

for ($y= $x-1; $y >=0; $y--) {
        print OUT $array[$y];
}

close(OUT);
```

C. LOGSEPARATOR.PL

```
#!      perl -w
#       Author:        Chandan Singh Negi
#       Description:   A short script to create multiple log files, one for each mib to
#                      facilitate the forming of RRDs
#       Usage:         perl –w logseparator.pl logfilename


my($file) = $ARGV[0];
my($time,$ipadd,$mib_n_value,$mib,$count);
my(@array);
my($FH);
my($no,$x) =0;
my($max_no_of_mibs);

open(LOG,"<$file") or die "could not open file";
while(<LOG>) {
        if ($_ !~ /^\n/){
                chomp;
                ($time,$ipadd,$mib_n_value) = split /:/;
                print "min_n_value = $mib_n_value\n";
                ($mib,$count) = split / = /, $mib_n_value;
                ($f,$m,$l) = split /\./, $mib;
                print "The value of log file name is $m.log\n";
                sleep 1;
                $array[$no] = $m.".log";
                print "The value of $no element in array is $array[$no]\n";
                $no++;
        }
      else {last;}
}
close(LOG);
$max_no_of_mibs = $no;
print "No. of mibs = $max_no_of_mibs\n";
sleep 1;

print "Splitting log files \n";

for ($x = 1; $x <= $max_no_of_mibs; $x++) {
#       open(OUT."$x", ">". "$x.log") || die "couldn't open $x.log \n";
        open(OUT."$x", ">". $array[$x-1]) || die "couldn't open $x.log \n";
}

$x = 1;
open(LOG,"<$file") or die "could not open file";
while(<LOG>) {
        if ($_ !~ /^\n/) {
                chomp;
                if ($x > $max_no_of_mibs) { $x = 1; }
                ($time,$ipadd,$mib_n_value) = split /:/;
                ($mib,$count) = split / = /, $mib_n_value;

                print "Updating $x.log\n";
                $FH = OUT."$x";
                print $FH "$time:$count\n";
                $x++;
        }
}
close(LOG);

for ($x = 1; $x <= $max_no_of_mibs; $x++) {
      $FH = OUT."$x";
        close($FH);
}
```

D. GRAPHER.PL

```
#!      perl -w
#       Author:         Chandan Singh Negi
#       Description:    A short perl script to create RRDs and Graphs. Uses perl::RRD
#                       modules
#       Usage:          perl –w grapher.pl miblogfilename

use RRDs;
use strict;
```

```
my($file) = $ARGV[0];
my($start_time,$end_time,$time,$count,$fname,$fext,$data,$x,$t);
my($ERROR);

($fname,$fext) = split /\./, $file;

print "The file is $file\n";
print "The filename is $fname\n";
print "The file extension is $fext\n";

open(LOG,"<$file") or die "could not open file";
my($cnt)=1;
while(<LOG>) {
        chomp;
        ($time,$count) = split /:/;
        if ($cnt <2) { $start_time = $time-10;}
        $end_time = $time;
        $cnt++;
}
close(LOG);

print "Creating Round-Robin-Database $fname.rrd \n";

RRDs::create ($fname.".rrd","--start",$start_time, "--step",5,
                "DS:a:COUNTER:10:U:U",
                "RRA:AVERAGE:0.5:1:600");
$ERROR = RRDs::error;
die "$0: unable to create '$fname.'.rrd: $ERROR\n" if $ERROR;


print "Updating $fname.rrd\n";

open(LOG,"<$file") or die "could not open file";
while(<LOG>) {
        chomp;
        RRDs::update ($fname.".rrd", $_);
        if ($ERROR = RRDs::error) {
                die "$0: unable to update $fname.rrd: $ERROR\n";
        };
}
close(LOG);

print " start time = $start_time\n";
print " end time = $end_time\n";

print "Making graphs\n";
```

```
#	Change the title name of the graph as desired.

RRDs::graph	("$fname.gif","--start",	$start_time,	"--end",	$end_time,"--
title","TFN_Syn_Flood - Rate of change of $fname with time",
	"--vertical-label","$fname(/s)", "DEF:mibvalue=$fname.rrd:a:AVERAGE",
	"LINE1:mibvalue#FF0000:\rAverage change in $fname per second");
if ($ERROR = RRDs::error) {
	die "$0: unable to draw graph $fname-AVG.gif: $ERROR\n";
};
```

LIST OF REFERENCES

[BARL- 00] J. Barlow and W. Thrower, "TFN2K – an analysis," Feb. 2000, http://packetstorm.securify.com/distributed/TFN2k_Analysis.htm

[BELL – 00] S. Bellovin, "Distributed Denial of Service attacks," Feb.2000, http://www.research.att.com/~smb/talks

[BELL –00-M] S. Bellovin. ICMP traceback messages, March 2000. Internet Draft: draft-ietf-itrace-00.txt

[CISC] CP Intercept - http://www.cisco.com/univercd/cc/td/doc/product/software/ios112/intercpt.htm

[CWRT –00-01] CERT Coordination Center, Cert Advisories: "CA-2000-01 denial-of-service developments," http://www.cert.org/advisories/CA-2000-01.html

[DITT – 99] D. Dittrich, " The Dos project's 'trinoo' distributed denial of service attack tool," Oct. 1999, " The 'Stacheldraht' distributed denial of service attack tool," Dec. 1999, " The 'Tribe Flood Network' distributed denial of service attack tool," Oct. 1999, http://www.washington.edu/People/dad

[DITT – 00] S. Dietrich, D. Dittrich and N. Long, " An analysis of the 'Shaft' distributed denial of service tool," Mar. 2000, http://www.adelphi.edu/~spock/shaft_analysis.txt

[EHAT – 00] http://www.ehatchery.com/press/ehatchery_09-13-00.html

[FERG – 98] P. Ferguson and D. Senie, " Network ingress filtering : Defeating denial of service attacks which employ ip source address spoofing," RFC 2267, Jan 1998.

[JOAO – 01] Joao B. D. Cabrera, Lundy Lewis, Xinzhou Qin, Wenke Lee, Ravi K. Prasanth, B. Ravichandran and Raman K. Mehra – Proactive Detection of Distributed Denial of Service Attacks using MIB Traffic Variables – A Feasibility Study – Proceedings of the 7th IFIP/IEEE international Symposium on Integrated Network Management, Seattle, WA.

[LAU –00] F. Lau, S. H. Rubin, M. H. Smith, and Lj. Trajkovic, "Distributed denial of service attacks" *Proc. 2000 IEEE Int. Conf. On Systems, Man, and Cybernetics*, Nashville, TN, Oct. 2000, pp. 2275-2280.

[NIPC – 00] http ://www.nipc.gov/warnings/advisories/2000/00-063.htm

[PARK-00] Kihong Park, Heejo Lee – A Proactive Approach to Distributed DoS Attack Prevention using Route-Based Packet Filtering.

[SAVA-00] S. Savage, D. wetherall, A. Karlin, and T. Anderson. Practical network support for IP traceback. – Proc of ACM SIGCOMM, pages 295-306- Aug 2000.

[SONG –01] Dawn Xiaodong Song and Adrian Perrig – Computer Science Departmentn – Advanced and Authenticated Marking Schemes for IP Traceback – IEEE INFOCOM 2001.

[STALL] SNMP, SNMPv2, SNMPv3, and RMON 1 and 2 Third Edition – William Stallings.

[STROT – 00] Elizabeth Strother, NCSU , Aug 29 2000, Denial of Service Protection – The Nozzle - http://ieeexplore.ieee.org/iel5/7224/19469/00898855.pdf

[STON-99] Robert Stone – CenterTrack - An IP Overlay Network for Tracking DoS Floods. –http://www.nanog.org/mtg-9910/robert.html

[TOBI-00] The RRDTool home page - http://people.ee.ethz.ch/~oetiker/webtools/rrdtool

THIS PAGE INTENTIONALLY LEFT

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Fort Belvoir, Virginia 22060-6218

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Prof. Dan Boger
   Naval Postgraduate School
   Monterey, CA 93943

4. Prof. Carl R. Jones
   Naval Postgraduate School
   Monterey, CA 93943

5. Prof. Alex Bordetsky
   Naval Postgraduate School
   Monterey, CA 93943

6. Lieutenant Commander Christopher S. Eagle
   Naval Postgraduate School
   Monterey, CA 93943

7. Paul Clark
   Naval Postgraduate School
   Monterey, CA 93943

8. The Chief of the Naval Staff
   (for DNT/DNI/ACOM(IT&S)/ACNS(IW(OPS)/Oi/C,IW Cell/DOS(W)/DOS(L))
   Naval Headquarters
   South Block
   New Delhi - 110 011, India

9. Naval Attaché
Embassy Of India
2107 Massachusetts Avenue, NW
Washington DC – 20008

10. The Commanding Officer
INS Valsura
Jamnagar
Gujarat - 361 001, India

11. The Commanding Officer
INS Shivaji
Tiger's Leap
Lonavala - 410 402
Pune, Maharashtra India

12. Chandan Singh Negi